THE NAS PARALLEL BENCHMARKS

edited by

David Bailey[1], John Barton[2], Thomas Lasinski[1], and Horst Simon[3]

Report RNR-91-002, January 1991

NAS Systems Division
Applied Research Branch
NASA Ames Research Center, Mail Stop T045-1
Moffett Field, CA 94035

January 28, 1991

Abstract. A new set of benchmarks has been developed for the performance evaluation of highly parallel supercomputers. These benchmarks consist of a set of kernels, the "Parallel Kernels", and a simulated application benchmark. Together they mimic the computation and data movement characteristics of large scale computational fluid dynamics (CFD) applications.

The principal distinguishing feature of these benchmarks is their "pencil and paper" specification — all details of these benchmarks are specified only algorithmically. In this way many of the difficulties associated with conventional benchmarking approaches on highly parallel systems are avoided.

---

[1]The author is a member of the NAS Applied Research Branch

[2]The author is a member of the NAS Development Branch

[3]The author is an employee of Computer Sciences Corporation. This work is supported through NASA Contract NAS 2-12961.

THE NAS PARALLEL BENCHMARKS

edited by
David Bailey[1], John Barton[2], Thomas Lasinski[1], and Horst Simon[3]
Numerical Aerodynamic Simulation (NAS) Systems Division
NASA Ames Research Center, Mail Stop T045-1
Moffett Field, CA 94035

---

[1]The author is a member of the NAS Applied Research Branch
[2]The author is a member of the NAS Development Branch
[3]The author is an employee of Computer Sciences Corporation. This work is supported through NASA Contract NAS 2-12961.

Abstract

A new set of benchmarks has been developed for the performance evaluation of highly parallel supercomputers. These benchmarks consist of a set of kernels, the "Parallel Kernels", and a simulated application benchmark. Together they mimic the computation and data movement characteristics of large scale computational fluid dynamics (CFD) applications.

The principal distinguishing feature of these benchmarks is their "pencil and paper" specification — all details of these benchmarks are specified only algorithmically. In this way many of the difficulties associated with conventional benchmarking approaches on highly parallel systems are avoided.

Chapter 1

GENERAL REMARKS

by D. Bailey[1], D. Browning[3], R. Carter[3], and H. Simon[3]

1.1 Introduction

The Numerical Aerodynamic Simulation (NAS) Program located at NASA Ames Research Center is a large scale effort whose goal is to advance the state of computational aerodynamics. Specifically, NAS desires (see [3], page 3) "to provide the Nation's aerospace research and development community by the year 2000 a high-performance, operational computing system capable of simulating an entire aerospace vehicle system within a computing time of one to several hours." The successful solution of this "Grand Challenge" problem will require the development of computer systems which can perform the required complex scientific computations at a sustained rate nearly one thousand times greater than current supercomputers can now achieve. The architecture of computer systems able to achieve this level of performance will likely be dissimilar to the shared memory multiprocessing supercomputers of today. While no consensus exists on what such an architecture will resemble, it is likely that the architecture will at minimum require thousands of processors computing in parallel. The solution of Grand Challenge problems will require highly parallel computer systems.

---

[1]The author is a member of the NAS Applied Research Branch

[3]The author is an employee of Computer Sciences Corporation. This work is supported through NASA Contract NAS 2-12961.

Highly parallel systems with computing power roughly equivalent to traditional shared memory multiprocessors exist today. Unfortunately, for various reasons, the performance evaluation of these systems on comparable types of scientific computations is extraordinarily difficult. Little relevant data is available for the performance of algorithms of interest to the computational aerophysics community on many currently available parallel systems. Benchmarking and performance evaluation of such systems has not kept pace with advances in hardware, software and algorithms. In particular, there is as yet no generally accepted benchmark program or even a benchmark strategy for these systems.

The popular "kernel" benchmarks that have been used for traditional vector supercomputers, such as the Livermore Loops [14], the LINPACK benchmark [9, 10] and the original NAS Kernels [7], are clearly inappropriate for the performance evaluation of highly parallel machines. In addition to the dubious meaning of the actual performance numbers, the computation and memory requirements of these programs do not do justice to the vastly increased capabilities of the new parallel machines, particularly those systems that will be available by the mid-1990s.

On the other hand, a full scale scientific application is similarly unsuitable. First of all, porting a large program to a new parallel computer architecture requires a major effort, and it is usually hard to justify a major research task simply to obtain a benchmark number. For that reason we believe that the otherwise very successful PERFECT Club benchmark [13] is not suitable for parallel systems. This is demonstrated by only very sparse performance results for parallel machines in the recent reports [16, 17, 8].

Alternatively, an application benchmark could assume the availability of automatic software tools for transforming "dusty deck" Fortran into efficient parallel code on a variety of systems. However, such tools do not exist today, and we frankly doubt that they will ever exist across a wide range of architectures.

Some other considerations for the development of a meaningful benchmark for a highly parallel supercomputer are the following:

- Advanced parallel systems frequently require new algorithmic and software approaches, and these new methods are often quite different from the conventional methods implemented in Fortran code for a sequential or vector machine.

- Benchmarks must be "generic" and should not favor any particular parallel architecture. This requirement precludes the usage of any architecture-specific code, such as message passing code.
- The correctness of results and performance figures must be easily verifiable. This requirement implies that both input and output data sets must be kept very small. It also implies that the nature of the computation and the expected results must be specified in great detail.
- The memory size and run time requirements must be easily adjustable to accommodate new systems with increased power.
- The benchmark must be readily distributable.

In our view, the only benchmarking approach that satisfies all of these constraints is a "paper and pencil" benchmark. The idea is to specify a set of problems only algorithmically. Even the input data must be specified only on paper. Naturally, the problem has to be specified in sufficient detail that a unique solution exists, and the required output has to be brief yet detailed enough to certify that the problem has been solved correctly. The person or persons implementing the benchmarks on a given system are expected to solve the various problems in the most appropriate way for the specific system. The choice of data structures, algorithms, processor allocation and memory usage are all (to the extent allowed by the specification) left open to the discretion of the implementer. Some extension of Fortran-77 is required, and reasonable limits are placed on the usage of assembly code and the like, but otherwise programmers are free to utilize language constructs that give the best performance possible on the particular system being studied.

To this end, we have devised a number of relatively simple "kernels", which are specified completely in part 2 of this document. However, kernels alone are insufficient to completely assess the performance potential of a parallel machine on "real" scientific applications. The chief difficulty is that a certain data structure may be very efficient on a certain system for one of the isolated kernels, and yet this data structure would be inappropriate if incorporated into a larger application. In other words, the performance of a real computational fluid dynamics (CFD) application on a parallel system is critically dependent on data motion between computational kernels. Thus

we consider the complete reproduction of this data movement to be of critical importance in a benchmark.

Our benchmark set therefore consists of two major components: the parallel kernels and a simulated application. The simulated application benchmark combines several computations in a manner that resembles the actual order of execution in certain important CFD application codes. This is discussed in more detail in chapter 3.

We feel that this benchmark set successfully addresses many of the problems associated with benchmarking parallel machines. Although we do not claim that this set is typical of all scientific computing, it is based on the key components of several large aeroscience applications used by scientists on supercomputers at NASA Ames Research Center. These benchmarks will be used by the Numerical Aerodynamic Simulation (NAS) Program to evaluate the performance of parallel computers.

1.2 Benchmark Rules

1.2.1 Definitions

In the following, the term "processor" is defined as a hardware unit capable of integer and floating point computation. The "local memory" of a processor refers to randomly accessible memory that can be accessed by that processor in less than one microsecond. The term "main memory" refers to the combined local memory of all processors. This includes any memory shared by all processors that can be accessed by each processor in less than one microsecond. The term "mass storage" refers to non-volatile randomly accessible storage media that can be accessed by at least one processor within forty milliseconds. A "processing node" is defined as a hardware unit consisting of one or more processors plus their local memory, which is logically a single unit on the network that connects the processors.

The term "computational nodes" refers to those processing nodes primarily devoted to high-speed floating point computation. The term "service nodes" refers to those processing nodes primarily devoted to system operations, including compilation, linking and communication with external computers over a network.

1.2.2 General Rules

Although the details of each parallel implementation of these benchmarks will differ for each architecture, the implementation is required to be expressed in a superset of the Fortran 77 language. This requirement stems from the observation that the commonly used languages by the scientific computing community, and the commonly implemented parallel computer languages, are based on Fortran 77. The rules proposed here provide a very general interpretation of 'Fortran' and intend to accommodate almost all extensions to Fortran as well as constructs which faciliate the implementation of the benchmark programs on a parallel machine, and which are currently in use in the scientific computing community. Thus in performing the NAS Parallel Benchmarks, the following rules must be complied with:

- All floating point operations must be performed using 64 bit floating point arithmetic.
- All benchmarks must be coded in Fortran-77, with certain approved extensions.
- Any extension of Fortran-77 that is in the Fortran-90 draft dated June 1990 or later is allowed [1].
- Any extension of Fortran-77 that is in the Parallel Computer Fortran (PCF) draft dated March 1990 or later is allowed [4].
- Any Fortran extension or library routine that is employed in any of the benchmarks must be supported by the vendor and available to all users.
- Non-Fortran subprograms or library routines may only perform certain functions, as indicated on the next section.
- All rules apply equally to subroutine calls, Fortran extensions and compiler directives (i.e. special comments).

1.2.3 Allowable Fortran Extensions and Library Routines

The following Fortran extensions and library routines are also permitted:

- Constructs that indicate sections of code that can be executed in parallel or loops that can be distributed among different computational nodes.
- Constructs that specify the allocation and organization of data among or within computational nodes.
- Constructs that communicate data between processing nodes.
- Constructs that communicate data between the computational nodes and service nodes.
- Constructs that rearrange data stored in multiple computational nodes, including constructs to perform indirect addressing and array transpositions.
- Constructs that synchronize the action of different computational nodes.
- Constructs that initialize for a data communication or synchronization operation that will be performed or completed later.
- Constructs that perform high-speed input or output operations between main memory and the mass storage system.
- Constructs that perform any of the following array reduction operations on an array either residing within a single computational node or distributed among multiple nodes: $+, \times$, `MAX`, `MIN`, `AND`, `OR`, `XOR`.
- Constructs that combine communication between nodes with one of the operations listed in the previous item.
- Constructs that perform any of the following computational operations on arrays either residing within a single computational node or distributed among multiple nodes: matrix-matrix multiplication, matrix-vector multiplication and one-dimensional, two-dimensional or three-dimensional fast Fourier transforms. Any such construct that is employed must be available as part of the system software and must be callable with general array dimensions.

1.3 Sample Codes

The intent of the NAS Parallel Benchmarks report is to completely specify the computation to be carried out in chapters 2 and 3. However, the developers of the benchmark are aware of the difficulty of generating a complete sets of programs from scratch from just the problem description. A significant portion of the benchmark programs also involves the generation of input data. Even though this report contains in principle all the necessary information which completely describes the benchmark problems, and which is necessary to generate benchmark programs, the availability of sample codes will significantly shorten the development time on a parallel machine. Furthermore sample computer programs will reduce the likelihood of ambiguities as to exactly what problem is to be solved.

In order to circumvent these difficulties, and to aid the benchmarking specialist, a set of Fortran 77 computer programs implementing each benchmark has been written. The programs are to be considered examples of how the problems may be solved, rather than requirements. As an aid to implementation, the programs actually solve scaled down versions of the benchmark problems. Instructions are supplied as comments in the source code on how to scale up the program parameters to the actual full size benchmark specification. These programs are available on request on a Macintosh floppy disk from the Applied Research Branch, NAS Systems Division, Mail Stop T045-1, NASA Ames Research Center, Moffett Field, CA 94035, attn: NAS Parallel Benchmark Codes.

The floppy disk contains the program sources, ReadMe files, data files, and reference output data for correct implementations of the benchmark problems. These codes have been validated on a number of computer systems ranging from conventional workstations to supercomputers.

For reference we list in table 1.1 the sample codes and the approximate resource requirements in the table below. Memory and CPU requirements of the sample codes are such that they can be solved easily on current generation workstations. Note that the unit "Mw" in tables 1.1 and 1.2. refers to one million 64 bit words. Also note that performance in MFLOPS is meaningless for the integer sort (IS) benchmark, and therefore not given. An explanation of the entries in the problem size column can be found in the corresponding sections describing the benchmarks.

Table 1.1: **NAS Parallel Benchmarks Sample Codes.** (Times and MFLOPS for one processor of the Cray Y-MP)

Benchmark code	Problem Size	Memory (Mw)	Time (sec)	MFLOPS
Embarrassingly parallel (EP)	2^{24}	0.1	11.6	120
Multigrid (MG)	32^3	0.1	0.1	128
Conjugate gradient (CG)	$\approx 10^5$	0.6	1.2	63
3-D FFT PDE (FT)	64^3	2.0	1.2	160
Integer sort (IS)	2^{16}	0.3	0.2	NA
LU solver (LU)	12^3	0.3	3.5	28
Pentadiagonal solver (SP)	12^3	0.2	7.2	24
Block tridiagonal solver (BT)	12^3	0.3	7.2	34

1.4 Submission of Benchmark Results

It must be emphasized again that the sample codes described in the section 1.3 are not the benchmark codes, but only implementation aids. For the actual benchmark the problems will have be scaled to larger problem sizes. How this is done is described in detail in the corresponding sections on the benchmarks. In table 1.2 as a summary the sizes, approximate times, and memory requirements of the actual benchmarks are listed.

The sizes of the current benchmarks were chosen such that an implementation on current generations of supercomputers is possible. The actual requirements at NAS are for much larger problems, with corresponding increases in memory sizes. Future releases of the benchmark will specify larger problem size.

The NAS Parallel Benchmark Group encourages submissions of benchmark results for the problems listed in table 1.2. Periodic publication of the submitted results is planned. Benchmark results should be submitted to the Applied Research Branch, NAS Systems Division, Mail Stop T045-1, NASA Ames Research Center, Moffett Field, CA 94035, attn: NAS Parallel Benchmark Results. A complete submission of results should include the following:

- Detailed description of the hardware and software configuration used for the benchmark.

Table 1.2: **NAS Parallel Benchmarks Problem Sizes.** (Times and MFLOPS for one processor of the Cray Y-MP)

Benchmark code	Problem Size	Memory (Mw)	Time (sec)	MFLOPS
Embarrassingly parallel (EP)	2^{28}	1	151	147
Multigrid (MG)	256^3	57	54	154
Conjugate gradient (CG)	$\approx 2 * 10^6$	12	22	70
3-D FFT PDE (FT)	$256 \times 256 \times 128$	59	39	192
Integer sort (IS)	2^{23}	26	21	NA
LU solver (LU)	64^3	8	344	189
Pentadiagonal solver (SP)	64^3	6	806	175
Block tridiagonal solver (BT)	64^3	6	923	192

- Description of the implementation, algorithmic techniques etc.
- Listing of the benchmark codes.
- Output listings from the benchmarks.

Chapter 2

THE KERNEL BENCHMARKS

by D. Bailey[1], E. Barszcz[1], L. Dagum[3], P. Frederickson[4], R. Schreiber[4], and H. Simon[3]

2.1 Overview

After an evaluation of a number of large scale CFD and computational aerosciences applications on the NAS supercomputers at NASA Ames, a number of kernels were selected for the benchmark. These were supplemented by some other kernels which are intended to test specific features of parallel machines. The following benchmark set was then assembled:

EP: An "embarrassingly parallel" kernel. It provides an estimate of the upper achievable limits for floating point performance, i.e. the performance without significant interprocessor communication.

MG: A simplified multigrid kernel. It requires highly structured long distance communication and tests both short and long distance data communication.

---

[1]The author is a member of the NAS Applied Research Branch

[3]The author is an employee of Computer Sciences Corporation. This work is supported through NASA Contract NAS 2-12961.

[4]The author is with RIACS. This work is supported by NAS Systems Division through Cooperative Agreement Number NCC 2-387.

CG: A conjugate gradient method is used to compute an approximation to the smallest eigenvalue of a large, sparse, symmetric positive definite matrix. This kernel is typical of unstructured grid computations in that it tests irregular long distance communication, employing unstructured matrix vector multiplication.

FT: A 3-D partial differential equation solution using FFTs. This kernel performs the essence of many "spectral" codes. It is a rigorous test of long-distance communication performance.

IS: A large integer sort. This kernel performs a sorting operation that is important in "particle method" codes. It tests both integer computation speed and communication performance.

These kernels involve substantially larger computations than previous kernel benchmarks, such as the Livermore Loops or Linpack, and therefore they are more appropriate for the evaluation of parallel machines. The Parallel Kernels in particular are sufficiently simple that they can be implemented on a new system without unreasonable effort and delay. Most importantly, as emphasized earlier, this set of benchmarks incorporates a new concept in performance evaluation, namely that only the computational task is specified, and that the actual implementation of the kernel can be tailored to the specific architecture of the parallel machine.

In this chapter the Parallel Kernel benchmarks are presented, and the particular rules for allowable changes are discussed. Future reports will describe implementations and benchmarking results on a number of parallel supercomputers.

2.2 Description of the Kernels

2.2.1 Kernel EP: An Embarrassingly Parallel Benchmark

by D. Bailey and P. Frederickson

Brief Statement of Problem:

Generate pairs of Gaussian random deviates according to a specific scheme described below and tabulate the number of pairs in successive square annuli.

Details:

Set $n = 2^{28}$, $a = 5^{13}$ and $s = 271828183$. Generate the pseudorandom floating point values r_j in the interval (0, 1) for $1 \le j \le 2n$ using the scheme described in section 2.3. Then for $1 \le j \le n$ set $x_j = 2r_{2j-1} - 1$ and $y_j = 2r_{2j} - 1$. Thus x_j and y_j are uniformly distributed on the interval $(-1, 1)$.

Next set $k = 0$. Then beginning with $j = 1$, test to see if $t_j = x_j^2 + y_j^2 \le 1$. If not, reject this pair and proceed to the next j. If this inequality holds, then set $k \leftarrow k + 1$, $X_k = x_j\sqrt{(-2\log t_j)/t_j}$ and $Y_k = y_j\sqrt{(-2\log t_j)/t_j}$, where log denotes the natural logarithm. Then X_k and Y_k are independent Gaussian deviates with mean zero and variance one. Approximately $n\pi/4$ pairs will be constructed in this manner. See ([12], p. 117) for additional discussion of this scheme for generating Gaussian deviates.

Finally, for $0 \le l \le 9$ tabulate Q_l as the count of the pairs (X_k, Y_k) that lie in the square annulus $l \le \max(|X_k|, |Y_k|) < l + 1$, and output the ten Q_l counts.

Verification Test:

Each of the ten Q_l counts must agree exactly with reference values. For this value of n, the reference counts are as follows:

l	Q_l
0	98257395
1	93827014
2	17611549
3	1110028
4	26536
5	245
6	0
7	0
8	0
9	0

Operations to be Timed:

All operations described above, including tabulation and output.

Computational Cost:

Approximately $n(45 + 12\pi)$ floating point operations. For this value of

n, this is 2.22×10^{10} floating point operations. This count is based on 19 floating point operations for each pseudorandom number, 12 for each square root and 25 for each logarithm evaluation.

Memory Requirement:

Minimal — storage is required only for the uniform pseudorandom numbers generated in a single batch.

Other Features:

- This problem is typical of many Monte-Carlo simulation applications.
- The only requirement for communication is the combination of the 10 sums from various processors at the end.
- Separate sections of the uniform pseudorandom numbers can be independently computed on separate processors. See section 2.3 for details.
- The smallest distance between a floating-point value and a nearby integer among the r_j, X_k and Y_k values is 3.2×10^{-11}, which is well above the achievable accuracy using 64 bit floating arithmetic on existing computer systems. Thus if a truncation discrepancy occurs, it implies a problem with the system hardware or software.

2.2.2 Kernel MG: A Simple 3D Multigrid Benchmark

by E. Barszcz and P. Frederickson

Brief Statement of Problem:

Four iterations of the V-cycle multigrid algorithm described below are used to obtain an approximate solution u to the discrete Poisson problem

$$\nabla^2 u = v$$

on a $256 \times 256 \times 256$ grid with periodic boundary conditions.

Details:

Set $v = 0$ except at the twenty points listed in table 2.1. where $v = \pm 1$. (These points were determined as the locations of the ten largest and ten smallest pseudorandom numbers generated as in Kernel FT.)

Table 2.1:

$v_{i,j,k}$	(i,j,k)				
-1.0	211,154, 98	102,138,112	101,156, 59	17,205, 32	92, 63,205
	199, 7,203	250,170,157	82,184,255	154,162, 36	223, 42,240
+1.0	57,120,167	5,118,175	176,246,164	45,194,234	212, 7,248
	115,123,207	202, 83,209	203, 18,198	243,172, 14	54,209, 40

Begin the iterative solution with $u = 0$. Each of the four iterations consists of the following two steps, in which $k = 8 = \log_2(256)$:

$$r = v - A\,u \qquad (\textit{evaluate residual})$$

$$u = u + M^k\,r \qquad (\textit{apply correction})$$

Here M^k denotes the V-cycle multigrid operator, defined in table 2.2. In this

Table 2.2:

z_k	=	$M^k r_k$	:	
	if $k > 1$			
		r_{k-1}	$= P\,r_k$	(restrict residual)
		z_{k-1}	$= M^{k-1} r_{k-1}$	(recursive solve)
		z_k	$= Q\,z_{k-1}$	(prolongate)
		r_k	$= r_k - A\,z_k$	(evaluate residual)
		z_k	$= z_k + S\,r_k$	(apply smoother)
	else			
		z_1	$= S\,r_1.$	(apply smoother)

definition A denotes the trilinear finite element discretization of the Laplacian ∇^2 normalized as indicated in table 2.3, where the coefficients of P, Q, and S are also listed.

In this table c_0 denotes the central coefficient of the 27-point operator, when these coefficients are arranged as a $3 \times 3 \times 3$ cube. Thus c_0 is the coefficient that multiplies the value at the gridpoint (i,j,k), while c_1 multiplies the six values at grid points which differ by one in exactly one index, c_2 multiplies the next closest twelve values, those that differ by one in exactly two indices, and c_3 multiplies the eight values located at grid points that

Table 2.3:

C	c_0	c_1	c_2	c_3
A	−8.0/3.0	0.0	1.0/6.0	1.0/12.0
P	1.0/2.0	1.0/4.0	1.0/8.0	1.0/16.0
Q	1.0	1.0/2.0	1.0/4.0	1.0/8.0
S	−3.0/8.0	+1.0/32.0	−1.0/64.0	0.0

differ by one in all three indices. The restriction operator P given in this table is the trilinear projection operator of finite element theory, normalized so that the coefficients of all operators are independent of level, and is half the transpose of the trilinear interpolation operator Q. The smoothing operator S specified in this table is chosen to work well with A.

Verification Test:

Evaluate the residual after four iterations of the V-cycle multigrid algorithm, and verify that its $\mathbf{L_2}$ norm

$$\|r\|_2 = [\,(\sum_{i,j,k} r_{i,j,k})/256^3\,]^{1/2}$$

agrees with the reference value

$$0.2433365309 \times 10^{-05}$$

within an absolute tolerance of 10^{-14}.

Timing:

Start the clock before evaluating the residual for the first time, and after initializing u and v. Stop the clock after evaluating the norm of the final residual, but before displaying or printing its value.

Computational Cost:

Approximately 6×10^9 floating point operations and 60 Mwords of memory are required in the most straightforward implementation. The operation count can be reduced by a factor of two at the cost of greater memory usage.

2.2.3 Kernel CG: Solving an Unstructured Sparse Linear System by the Conjugate Gradient Method

by R. Schreiber and H. Simon

Brief Statement of Problem:

In this benchmark, the power method is used to find an estimate of the smallest eigenvalue of a symmetric positive definite sparse matrix with a random pattern of nonzeros.

Details:

A is a symmetric, positive definite, sparse matrix generated by the program described below. The matrix is of order 14,000 and has 1,853,104 nonzero elements. In the following, A is the sparse matrix, lower case Roman letters are vectors, x_j is the j^{th} component of x, and lower case Greek letters are scalars. We denote by $||x||$ the Euclidean norm of a vector x, $||x|| = \sqrt{\sum_{i=1}^{14,000} x_i^2}$. All quantities are real.

The power method is to be implemented as follows:

$x = [1, 1, \ldots, 1]$;
$\zeta_2 = 0$;
$\zeta_1 = 0$;
$\zeta = 0$;
$it = 0$;
outer iteration:
do 15 times
    $it \leftarrow it + 1$
    Solve the system $Az = x$;
    $\zeta_2 = \zeta_1$;
    $\zeta_1 = \zeta$;
    $\zeta = \max_j |z_j|$;
    $x = \zeta^{-1} z$;
    Apply Aitken extrapolation to the last three iterates ζ, ζ_1, and ζ_2 to produce an improved approximation ζ' with the following formula
    $\zeta' = \zeta - \frac{(\zeta - \zeta_1)^2}{\zeta - 2\zeta_1 + \zeta_2}$

Print it, ζ', and $||r||$, the Euclidean norm of the last CG residual vector;
od

The values of ζ and ζ' are increasingly accurate approximations to the largest eigenvalue of A^{-1}, which is the reciprocal of the smallest eigenvalue of A. By using the formula of the above algorithm, Aitken extrapolation of a linearly convergent sequence $\{\zeta_n\}$ produces a more rapidly converging sequence $\{\zeta'_n\}$.

The solution z to the linear system of equations $Az = x$ is to be approximated using the conjugate gradient (CG) method. This method is to be implemented as follows:

$z = 0$;
$r = x$;
$\rho = r^T r$;
$p = r$;
do 25 times
    $q = Ap$;
    $\alpha = \rho/(p^T q)$;
    $z = z + \alpha p$;
    $\rho_0 = \rho$;
    $r = r - \alpha q$;
    $\rho = r^T r$;
    $\beta = \rho/\rho_0$;
    $p = r + \beta p$;
    od

Verification Test:

The program should print, at every outer iteration of the the power method, the iteration number it, the value of ζ', and the Euclidean norm $||r||$ of the residual vector at the last CG iteration (the vector r in the discussion of CG above).

The final value of ζ' printed by the program must agree with the reference value 0.101221137511 within a tolerance of 1.0×10^{-10}, i.e. $|\zeta' - 0.101221137511\ | \leq 1.0 \times 10^{-10}$.

Timing:

The reported time must be the time required to compute all 15 iterations and print the results, after the matrix is generated and downloaded into the parallel machine, and after the initialization of the starting vector x.

It is permissible initially to reorganize the sparse matrix data structure (`arow`, `acol`, `aelt`) which is produced by the matrix generation routine, to a data structure better suitable for the target machine. The original or the reorganized sparse matrix data structure can then be subsequently used in the conjugate gradient interation. Time spent in the initial reorganization of the data structure will *not* be counted towards the benchmark time.

It is also permissible to use several different data structures for the matrix A, keep multiple copies of the matrix A, or to write A to mass storage, and read it back in. However, the time for any data movements, which take place within the power method iterations (outer iteration) or within the conjugate gradient iterations (inner iteration), must be included in the reported time.

Computational Cost:

Generating the matrix requires 6.2 seconds on one processor of a Cray Y-MP, but as mentioned above this is not counted as part of the kernel for timing purposes. Approximately 9.1×10^9 floating point operations are required for the timed portion of the test.

Memory Requirement:

The storage requirement for the timed portion of this kernel is about 2 Mwords of 64 bit words of memory. The sparse matrix generation program requires additional workspace for about 6 million integer words.

Other Features:

The input sparse matrix A is generated by a Fortran 77 subroutine called **makea**, which is provided on the sample code disk described in section 1.3. In this program, the random number generator is initialized with $a = 5^{13}$ and $s = 314159265$. Then the subroutine **makea** is called to generate the matrix A. This program may not be changed.

In routine **makea** the matrix A is represented as follows:

`N (INTEGER)` — the number of rows and columns

`NZ (INTEGER)` — the number of nonzeros

A (REAL*8) — array of NZ nonzeros

IA (INTEGER) — array of NZ row indices. Element A(K) is in row IA(K) for all $1 \leq K \leq NZ$.

JA (INTEGER) — array of N+1 pointers to the beginnings of columns. Column J of the matrix is stored in positions JA(J) through JA(J+1)-1 of A and IA. JA(N+1) contains NZ+1.

The code generates the matrix as the weighted sum of N outer products of random sparse vectors x:

$$A = \sum_{i=1}^{N} \omega_i x x^T,$$

where the weights ω_i are a geometric sequence with $\omega_1 = 1$ and ratio chosen so that $\omega_N = 0.1$. The vectors x are chosen to have a few randomly placed nonzeros, each of which a sample from the uniform distribution on $(0, 1)$. Furthermore, the i^{th} element of x_i is set to $1/2$ to insure that A cannot be structurally singular. Finally, 0.1 is added to the diagonal of A. This results in a matrix whose condition number (the ratio of its largest eigenvalue to its smallest) is roughly 10. The number of randomly chosen elements of x is taken to be 11 ; the final number of nonzeros in A is 1,853,104 .

The data structures used are these. First, a list of triples (arow, acol, aelt) is constructed. Each of these represents an element in row $i = arow$, column $j = acol$ with value $a_{ij} = aelt$. When the arow and acol entries of two of these triples coincide, then the values in their aelt fields are added together in creating a_{ij}. The process of assembling the matrix data structures from the list of triples, including the process of adding coincident entries, is done by the subroutine sparse, which is called by makea and also provided. For examples and more details on this sparse data structure consult section 2.7 of the book by Duff, Erisman, and Reid [11].

2.2.4 Kernel FT: A 3-D FFT PDE Benchmark

by D. Bailey and P. Frederickson

Brief Statement of Problem:

Numerically solve a certain partial differential equation (PDE) using forward and inverse FFTs.

Details:

Consider the PDE

$$\frac{\partial u(x,t)}{\partial t} = \alpha \nabla^2 u(x,t)$$

where x is a position in 3-dimensional space. When a Fourier transform is applied to each side, this equation becomes

$$\frac{\partial v(z,t)}{\partial t} = -4\alpha\pi^2 |z|^2 v(z,t)$$

where $v(z,t)$ is the Fourier transform of $u(x,t)$. This has the solution

$$v(z,t) = e^{-4\alpha\pi^2|z|^2 t} v(z,0)$$

Now consider the discrete version of the original PDE. Following the above, it can be solved by computing the forward 3-D discrete Fourier transform (DFT) of the original state array $u(x,0)$, multiplying the results by certain exponentials, and then performing an inverse 3-D DFT. The forward DFT and inverse DFT of the $n_1 \times n_2 \times n_3$ array u are defined respectively as

$$F_{r,s,t}(u) = \sum_{l=0}^{n_3-1} \sum_{k=0}^{n_2-1} \sum_{j=0}^{n_1-1} u_{j,k,l} e^{-2\pi i jr/n_1} e^{-2\pi i ks/n_2} e^{-2\pi i lt/n_3}$$

$$F^{-1}_{r,s,t}(u) = \frac{1}{n_1 n_2 n_3} \sum_{l=0}^{n_3-1} \sum_{k=0}^{n_2-1} \sum_{j=0}^{n_1-1} u_{j,k,l} e^{2\pi i jr/n_1} e^{2\pi i ks/n_2} e^{2\pi i lt/n_3}$$

The specific problem to be solved in this benchmark is as follows. Set $n_1 = 256$, $n_2 = 256$, and $n_3 = 128$. Generate $2n_1 n_2 n_3$ 64-bit pseudorandom floating point values using the pseudorandom number generator in section 2.3, starting with the initial seed 314159265. Then fill the complex array $U_{i,j,k}$, $0 \le i < n_1$, $0 \le j < n_2$, $0 \le k < n_3$, with this data, where the first dimension varies most rapidly as in the ordering of a 3-D Fortran array. A single complex number entry of U consists of two consecutive pseudorandomly generated results. Compute the forward 3-D DFT of U, using a 3-D

fast Fourier transform (FFT) routine, and call the result V. Set $\alpha = 10^{-6}$ and set $t = 1$. Then compute

$$W_{i,j,k} = e^{-4\alpha\pi^2(\bar{i}^2+\bar{j}^2+\bar{k}^2)t} V_{i,j,k}$$

where $\bar{i}$ is defined as i for $0 \leq i < n_1/2$ and $i - n_1$ for $n_1/2 \leq i < n_1$. The indices $\bar{j}$ and $\bar{k}$ are similarly defined with n_2 and n_3. Then compute an inverse 3-D DFT on W, using a 3-D FFT routine, and call the result the array X. Finally, compute the complex checksum $\sum_{i=0}^{1023} X_{r,s,t}$ where $r = i \pmod{n_1}$, $s = 3i \pmod{n_2}$ and $t = 5i \pmod{n_3}$. After the checksum for this t has been output, increment t by one. Then repeat the above process, from the computation of W through the incrementing of t, until the step $t = N$ has been completed. In this benchmark, $N = 6$. The V array and the array of exponential terms for $t = 1$ need only be computed once. Note that the array of exponential terms for $t > 1$ can be obtained as the t-th power of the array for $t = 1$.

Any algorithm may be used for the computation of the 3-D FFTs mentioned above. One algorithm is as follows. Assume that the data in the input $n_1 \times n_2 \times n_3$ complex array A is organized so that for each j and k, all elements of the complex vector $(A_{i,j,k},\ 0 \leq i < n_1)$ are contained within a single processing node. First perform an n_1-point 1-D FFT on each of these $n_2 n_3$ complex vectors. Then transpose the resulting array into an $n_2 \times n_3 \times n_1$ complex array B. Next, perform an n_2-point 1-D FFT on each of the $n_3 n_1$ first-dimension complex vectors of B. Again note that each of the 1-D FFTs can be performed locally within a single node. Then transpose the resulting array into an $n_3 \times n_1 \times n_2$ complex array C. Finally, perform an n_3-point 1-D FFT on each of the $n_1 n_2$ first-dimension complex vectors of C. Then transpose the resulting array into an $n_1 \times n_2 \times n_3$ complex array D. This array D is the final 3-D FFT result.

Algorithms for performing an individual 1-D complex-to-complex FFT are well known and will not be presented here. Readers are referred to the references [5, 6, 15, 18, 19] for details. It might be noted that some of these FFTs are "unordered" FFTs, i.e. the results are not in the correct order but instead are scrambled by a bit-reversal permutation. Such FFTs may be employed if desired, but it should be noted that in this case the ordering of the exponential factors in the definition of $W_{i,j,k}$ above must be similarly scrambled in order to obtain the correct results. Also, the final result array

X may be scrambled, in which case the checksum calculation will have to be changed accordingly.

It should be noted that individual 1-D FFTs, array transpositions, and even entire 3-D FFT operations may be performed using vendor-supplied library routines. See sections 1.2.2 and 1.2.3 for details.

Operations to be Timed:

All of the above operations, including the checksum calculations, must be timed.

Verification Test:

The N complex checksums must agree with reference values to within one part in 10^{12}. For the parameter sizes specified above, the reference values are as follows:

t	Real Part	Imaginary Part
1	504.6735008193	511.4047905510
2	505.9412319734	509.8809666433
3	506.9376896287	509.8144042213
4	507.7892868474	510.1336130759
5	508.5233095391	510.4914655194
6	509.1487099959	510.7917842803

Computational Cost:

Approximately $n[58 + 6N + 5(N+1)\log_2 n]$ floating point operations, where $n = n_1 n_2 n_3$. For the parameter sizes specified above, this is 7.54×10^9 floating point operations. This count is based on 19 floating point operations for each pseudorandom number, $5m \log_2 m$ for each m-point complex FFT and 20 for each exponential function evaluation.

Memory Requirement:

Approximately $7n$ words (64 bit), where $n = n_1 n_2 n_3$. This assumes that a scratch array of the same size as the 3-D data array is required for the 3-D FFTs. For the parameter sizes specified above, this is 5.87×10^7 words.

Other Features:

- 3-D FFTs are a key part of certain CFD applications, notably large eddy turbulence simulations.

- The 3-D FFT steps require considerable communication for operations such as array transpositions.

2.2.5 Kernel IS: Parallel Sort Over Small Integers

by L. Dagum

Brief Statement of Problem:

Sort N keys in parallel. The keys are generated by the sequential key generation algorithm given below and initially must be uniformly distributed in memory. The initial distribution of the keys can have a great impact on the performance of this benchmark, and the required distribution is discussed in detail below.

Definitions:

A sequence of keys, $\{K_i \mid i = 0, 1, \ldots, N-1\}$, will be said to be *sorted* if it is arranged in non–descending order, i.e. $K_i \leq K_{i+1} \leq K_{i+2} \ldots$. The *rank* of a particular key in a sequence is the index value i that the key would have if the sequence of keys were sorted. *Ranking*, then, is the process of arriving at a rank for all the keys in a sequence. *Sorting* is the process of permuting the the keys in a sequence to produce a sorted sequence. If an initially unsorted sequence, $K_0, K_1, \ldots, K_{N-1}$ has ranks $r(0), r(1), \ldots, r(N-1)$, the sequence becomes sorted when it is rearranged in the order $K_{r(0)}, K_{r(1)}, \ldots, K_{r(N-1)}$. Sorting is said to be *stable* if equal keys retain their original relative order. In other words, a sort is stable only if $r(i) < r(j)$ whenever $K_{r(i)} = K_{r(j)}$ and $i < j$. Stable sorting is not required for this benchmark.

Memory Mapping:

The benchmark requires ranking an unsorted sequence of N keys. The initial sequence of keys will be generated in an unambiguous sequential manner described below. This sequence must be mapped into the memory of the parallel processor in one of the following ways depending on the type of memory system. In all cases, one key will map to one word of memory. Word size must be no less than 32 bits. Once the keys are loaded onto the memory system, they are not to be moved or modified except as required by the procedure described in the Procedure subsection.

Shared Global Memory All N keys initially must be stored in a contigu-

ous address space. If A_i is used to denote the address of the i^{th} word of memory, then the address space must be $[A_i, A_{i+N-1}]$. The sequence of keys, $K_0, K_1, \ldots, K_{N-1}$, initially must map to this address space as

$$A_{i+j} \longleftarrow MEM(K_j) \quad \text{for} \quad j = 0, 1, \ldots, N-1 \tag{2.1}$$

where $MEM(K_j)$ refers to the address of K_j.

Distributed Memory In a distributed memory system with p distinct memory units, each memory unit initially must store N_p keys in a contiguous address space, where

$$N_p = N/p. \tag{2.2}$$

If A_i is used to denote the address of the i^{th} word in a memory unit, and if P_j is used to denote the j^{th} memory unit, then $P_j \cap A_i$ will denote the address of the i^{th} word in the j^{th} memory unit. Some initial addressing (or "ordering") of memory units must be assumed and adhered to throughout the benchmark. Note that the addressing of the memory units is left completely arbitrary. If N is not evenly divisible by p, then memory units $\{P_j \mid j = 0, 1, \ldots, p-2\}$ will store N_p keys, and memory unit P_{p-1} will store N_{pp} keys, where now

$$N_p = \lfloor N/p + 0.5 \rfloor \tag{2.3}$$

$$N_{pp} = N - (p-1)N_p. \tag{2.4}$$

In some cases (in particular if p is large) this mapping may result in a poor initial load balance with $N_{pp} >> N_p$. In such cases it may be desirable to use p' memory units to store the keys, where $p' < p$. This is allowed, however the storage of the keys still must follow either equation 2.2 or equations 2.3–2.4 with p' replacing p. In the following we will assume N is evenly divisible by p. The address space in an individual memory unit must be $[A_i, A_{i+N_p-1}]$. If memory units are individually hierarchical, then N_p keys must be stored in a contiguous address space belonging to a single memory hierarchy and A_i then denotes the address of the i^{th} word in that hierarchy. The keys cannot be distributed amongst different memory hierarchies until after timing

begins. The sequence of keys, $K_0, K_1, \ldots, K_{N-1}$, initially must map to this distributed memory as

$$P_k \cap A_{i+j} \longleftarrow MEM(K_{kN_p+j}) \quad \text{for} \quad j = 0, 1, \ldots, N_p - 1 \quad \text{and} \quad k = 0, 1, \ldots, p - 1 \quad (2.5)$$

where $MEM(K_{kN_p+j})$ refers to the address of K_{kN_p+j}. If N is not evenly divisible by p, then the mapping given above must be modified for the case where $k = p - 1$ as

$$P_{p-1} \cap A_{i+j} \longleftarrow MEM(K_{(p-1)N_p+j}) \quad \text{for} \quad j = 0, 1, \ldots, N_{pp} - 1. \quad (2.6)$$

Hierarchical Memory All N keys initially must be stored in an address space belonging to a single memory hierarchy which will here be referred to as the *main memory*. Note that any memory in the hierarchy which can store all N keys may be used for the initial storage of the keys, and the use of the term "main memory" in the description of this benchmark should not be confused with the more general definition of this term in section 1.2.1. The keys cannot be distributed amongst different memory hierarchies until after timing begins. The mapping of the keys to the main memory must follow one of either the shared global memory or the distributed memory mappings described above.

The benchmark requires computing the rank of each key in the sequence. The mappings described above define the initial ordering of the keys. For shared global and hierarchical memory systems, the same mapping must be applied to determine the correct ranking. For the case of a distributed memory system, it is permissible for the mapping of keys to memory at the end of the ranking to differ from the initial mapping *only* in the following manner: *the number of keys mapped to a memory unit at the end of the ranking may differ from the initial value,* N_p. It is expected, in a distributed memory machine, that good load balancing of the problem will require changing the initial mapping of the keys and for this reason a different mapping may be used at the end of the ranking. If N_{p_k} is the number of keys in memory

unit P_k at the end of the ranking, then the mapping which must be used to determine the correct ranking is given by

$$P_k \cap A_{i+j} \longleftarrow MEM(r(kN_{p_k} + j)) \qquad \begin{array}{ll} \text{for} & j = 0, 1, \ldots, N_{p_k} - 1 \\ \text{and} & k = 0, 1, \ldots, p - 1 \end{array} \quad (2.7)$$

where $r(kN_{p_k} + j)$ refers to the rank of key $K_{kN_{p_k}+j}$. Note, however, this does not imply that the keys, once loaded into memory, may be moved. Copies of the keys may be made and moved, but the original sequence must remain intact such that each time the ranking process is repeated (Step 4 of the Procedure) the original sequence of keys exists (except for the two modifications of Step 4a) and the same algorithm for ranking is applied. Specifically, knowledge obtainable from the communications pattern carried out in the first ranking cannot be used to speed up subsequent rankings and each iteration of Step 4 should be completely independent of the previous iteration.

Key Generation Algorithm:

The algorithm for generating the keys makes use of the pseudorandom number generator described in section 2.3. The keys will be in the range $[0, B_{max})$. Let r_f be a random fraction uniformly distributed in the range $[0, 1]$, and let K_i be the i^{th} key. The value of K_i is determined as

$$K_i \longleftarrow \lfloor B_{max}(r_{4i+0}+r_{4i+1}+r_{4i+2}+r_{4i+3})/4 \rfloor \quad \text{for} \quad i = 0, 1, \ldots, N-1. \quad (2.8)$$

Note that K_i must be an integer and $\lfloor \cdot \rfloor$ indicates truncation. Four *consecutive* pseudorandom numbers from pseudorandom number generator must be used for generating each key. All operations before the truncation must be performed in 64-bit double precision. The random number generator must be initialized with $s = 314159265$ as a starting seed.

Partial Verification Test:

Partial verification is conducted for each ranking performed. Partial verification consists of comparing a particular subset of ranks with the reference values. The subset of ranks and the reference values are given in table 2.5 of the Specifications subsection. Note that the subset of ranks is selected to be

invariant to the ranking algorithm (recall that stability is not required in the benchmark). This is accomplished by selecting for verification only the ranks of unique keys. If a key is unique in the sequence (i.e. there is no other equal key), then it will have a unique rank despite an unstable ranking algorithm. The memory mapping described in the Memory Mapping subsection must be applied.

Full Verification Test:

Full verification is conducted after the last ranking is performed. Full verification requires the following:

1. Rearrange the sequence of keys, $\{K_i \mid i = 0, 1, \ldots, N-1\}$, in the order $\{K_j \mid j = r(0), r(1), \ldots, r(N-1)\}$, where $r(0), r(1), \ldots, r(N-1)$ is the last computed sequence of ranks.

2. For every K_i from $i = 0 \ldots N-2$ test that $K_i \leq K_{i+1}$.

If the result of this test is true, then the keys are in sorted order. The memory mapping described in the Memory Mapping subsection must be applied.

Procedure:

1. In a scalar sequential manner and using the key generation algorithm described above, generate the sequence of N keys.

2. Using the appropriate memory mapping described above, load the N keys into the memory system.

3. Begin timing.

4. Do, for $i = 1$ to I_{max}

   (a) Modify the sequence of keys by making the following two changes:

$$K_i \longleftarrow i \tag{2.9}$$

$$K_{i+I_{max}} \longleftarrow (B_{max} - i) \tag{2.10}$$

   (b) Compute the rank of each key.

   (c) Perform the partial verification test described above.

5. End timing.

6. Perform full verification test described above.

Computational Cost:

On a sequential machine, integer sorting has a time complexity of $O(N)$. For each ranking, the sequential algorithm has $2N + B_{max}$ arithmetic operations (integer add or subtract), and $7N + 4B_{max}$ memory references. No floating point operations are required.

Specifications:

The specifications given in table 2.4 shall be used in the benchmark. Two sets of values are given. The *Full Scale* values are the values to be used in the actual benchmark as described in section 1.4. However, for development purposes, the *Sample Code* values as described in section 1.3 may be used.

Parameter	Full Scale	Sample Code
N	2^{23}	2^{16}
B_{max}	2^{19}	2^{11}
seed	314159265	314159265
I_{max}	10	10

Table 2.4: Parameter values to be used for benchmark.

For partial verification, the reference values given in table 2.5 are to be used. In this table, $r(j)$ refers to the rank of K_j and i is the iteration of Step 4 of the Procedure. Again two sets of values are given, the *Full Scale* set being for the actual benchmark and the *Sample Code* set being for development purposes. It should be emphasized that the benchmark measures the performance based on use of the *Full Scale* values, and the *Sample Code* values are given only as a convenience to the implementor. Also to be supplied to the implementor is Fortran 77 source code for the sequential implementation of the benchmark using the *Sample Code* values and with partial and full verification tests.

Rank	Full Scale	Sample Code
$r(2112377)$	$104+i$	$0+i$
$r(662041)$	$17523+i$	$18+i$
$r(5336171)$	$123928+i$	$346+i$
$r(3642833)$	$8288932-i$	$64917-i$
$r(4250760)$	$8388264-i$	$65463-i$

Table 2.5: Values to be used for partial verification.

2.3 A Pseudorandom Number Generator for the Parallel NAS Kernels

by D. Bailey

Suppose that n uniform pseudorandom numbers are to be generated. Set $a = 5^{13}$ and let $x_0 = s$ be a specified initial "seed", i.e. an integer in the range $0 < s < 2^{46}$. Generate the integers x_k for $1 \le k \le n$ using the linear congruential recursion

$$x_{k+1} = ax_k \pmod{2^{46}}$$

and return $r_k = 2^{-46}x_k$ as the results. Thus $0 < r_k < 1$, and the r_k are very nearly uniformly distributed on the unit interval. See ([12], beginning on p. 9) for further discussion of this type of pseudorandom number generator.

Note that any particular value x_k of the sequence can be computed directly from the initial seed s by using the binary algorithm for exponentiation, taking remainders modulo 2^{46} after each multiplication. To be specific, let m be the smallest integer such that $2^m > k$, set $b = s$ and $t = a$. Then repeat the following for i from 1 to m:

$$\begin{aligned} j &\leftarrow k/2 \\ b &\leftarrow bt \pmod{2^{46}} \qquad \text{if } 2j \neq k \\ t &\leftarrow t^2 \pmod{2^{46}} \\ k &\leftarrow j \end{aligned}$$

The final value of b is $x_k = a^k s \pmod{2^{46}}$. See ([12], p. 442) for further discussion of the binary algorithm for exponentiation.

The operation of multiplying two large integers modulo 2^{46} can be implemented using 64 bit floating point arithmetic by splitting the arguments into two words with 23 bits each. To be specific, suppose one wishes to compute $c = ab \pmod{2^{46}}$. Then perform the following steps, where int denotes the greatest integer:

$$\begin{aligned}
a_1 &\leftarrow \text{int}\,(2^{-23}a) \\
a_2 &\leftarrow a - 2^{23}a_1 \\
b_1 &\leftarrow \text{int}\,(2^{-23}b) \\
b_2 &\leftarrow b - 2^{23}b_1 \\
t_1 &\leftarrow a_1 b_2 + a_2 b_1 \\
t_2 &\leftarrow \text{int}\,(2^{-23}t_1) \\
t_3 &\leftarrow t_1 - 2^{23}t_2 \\
t_4 &\leftarrow 2^{23}t_3 + a_2 b_2 \\
t_5 &\leftarrow \text{int}\,(2^{-46}t_4) \\
c &\leftarrow t_4 - 2^{46}t_5
\end{aligned}$$

An implementation of the complete pseudorandom number generator algorithm using this scheme produces the same sequence of results on any system that satisfies the following requirements:

- The input multiplier a and the initial seed s, as well as the constants 2^{23}, 2^{-23}, 2^{46} and 2^{-46}, can be represented exactly as 64 bit floating point constants.
- The truncation of a nonegative 64 bit floating point value less than 2^{24} is exact.
- The addition, subtraction and multiplication of 64 bit floating point values, where the arguments and results are nonnegative whole numbers less than 2^{47}, produce exact results.
- The multiplication of a 64 bit floating point value, which is a nonnegative whole number less than 2^{47}, by the 64 bit floating point value 2^{-m}, $0 \leq m \leq 46$, produces an exact result.

These requirements are met by virtually all scientific computers in use today. Any system based on the IEEE-754 floating point arithmetic standard [2] easily meets these requirements using double precision. However, it should be noted that obtaining an exact power of two constant on some systems requires a loop rather than merely an assignment statement with **.

Other Features:

- The period of this pseudorandom number generator is $2^{44} = 1.76 \times 10^{13}$, and it passes all reasonable statistical tests.
- This calculation can be vectorized on vector computers by generating results in batches of size equal to the hardware vector length.
- By using the scheme described above for computing x_k directly, the starting seed of a particular segment of the sequence can be quickly and independently determined. Thus numerous separate segments can be generated on separate processors of a multiprocessor system.
- Once the IEEE-754 floating point arithmetic standard gains universal acceptance among scientific computers, the radix 2^{46} can be safely increased to 2^{52}, although the scheme described above for multiplying two such numbers must be correspondingly changed. This will increase the period of the pseudorandom sequence by a factor of 64, to approximately 1.13×10^{15}.

Bibliography

[1] *Draft proposed Fortran 90 ANSI Standard X3J11.159-1989.* American National Standards Institute, 1430 Broadway, New York, NY, 10018, 1990.

[2] *IEEE Standard for Binary Floating Point Numbers, ANSI/IEEE Standard 754-1985.* IEEE, New York, 1985.

[3] *Numerical Aerodynamic Simulation Program Plan.* NAS Systems Division, NASA Ames Research Center, October 1988.

[4] *PCF Fortran Extensions – Draft Document, Revision 2.11.* Parallel Computing Forum(PCF), c/o Kuck and Associates, 1906 Fox Drive, Champaign, Illinois 61820, March 1990.

[5] R. C. Agarwal and J. W. Cooley. Fourier transform and convolution subroutines for the IBM 3090 Vector Facility. *IBM Journal of Research and Development*, 30:145 – 162, 1986.

[6] D. H. Bailey. A high-performance FFT algorithm for vector supercomputers. *International Journal of Supercomputer Applications*, 2:82 – 87, 1988.

[7] D.H. Bailey and J. Barton. *The NAS Kernel Benchmark Program.* Technical Report 86711, NASA Ames Research Center, Moffett Field, California, August 1985.

[8] G. Cybenko, L. Kipp, L. Pointer, and D. Kuck. *Supercomputer Performance Evaluation and the Perfect Benchmarks.* Technical Report 965, CSRD, Univ. of Illinois, Urbana, Illinois, March 1990.

[9] J. J. Dongarra. The LINPACK benchmark: an explanation. *SuperComputing*, 10 – 14, Spring 1988.

[10] J. J. Dongarra. *Performance of Various Computers Using Standard Linear Equations Software in a Fortran Environment.* Technical Report MCSRD 23, Argonne National Laboratory, March 1988.

[11] I.S. Duff, A.M. Erisman, and J.K. Reid. *Direct Methods for Sparse Matrices.* Clarendon Press, Oxford, 1986.

[12] D. E. Knuth. *The Art of Computer Programming, Vol. 2.* Addison-Wesley, 1981.

[13] M. Berry et al. The Perfect Club benchmarks: effective performance evaluation of supercomputers. *The International Journal of Supercomputer Applications*, 3:5 – 40, 1989.

[14] F. H. McMahon. *The Livermore Fortran Kernels: A Computer Test of the numerical performance Range.* Technical Report UCRL - 53745, Lawrence Livermore National Laboratory, Livermore, California, December 1986.

[15] M. C. Pease. An adaptation of the fast Fourier transform for parallel processing. *Journal of the ACM*, 252 – 264, 1968.

[16] L. Pointer. *PERFECT Report 1.* Technical Report 896, CSRD, Univ. of Illinois, Urbana, Illinois, July 1989.

[17] L. Pointer. *PERFECT Report 2: Performance Evaluation for Costeffective Transformations.* Technical Report 964, CSRD, Univ. of Illinois, Urbana, Illinois, March 1990.

[18] P. N. Swarztrauber. FFT algorithms for vector computers. *Parallel Computing*, 1:45 – 63, 1984.

[19] P. N. Swarztrauber. Multiprocessor FFTs. *Parallel Computing*, 5:197 – 210, 1987.

Chapter 3

THE APPLICATION BENCHMARKS

by S.Weeratunga[3], E. Barszcz[1], R. Fatoohi[3], and V. Venkatakrishnan[3]

[3]The author is an employee of Computer Sciences Corporation. This work is supported through NASA Contract NAS 2-12961.

[1]The author is a member of the NAS Applied Research Branch

Chapter 3

A METHODOLOGY FOR BENCHMARKING SOME CFD KERNELS ON HIGHLY PARALLEL PROCESSORS

Sisira Weeratunga Eric Barszcz, Rod Fatoohi and V. Venkatakrishnan
Numerical Aerodynamic Simulation (NAS) Systems Division
NASA Ames Research Center, Mail Stop T-045-1
Moffett Field, CA 94035

Abstract

A collection of iterative PDE solvers embedded in a pseudo application program is proposed for the performance evaluation of CFD codes on highly parallel processors. The pseudo application program is stripped of complexities associated with real CFD application programs, thereby enabling a simpler description of the algorithms. However, it is capable of reproducing the essential computation and data motion characteristics of large scale, state of the art CFD codes. In this chapter, we present a detailed description of the pseudo application program concept. Preceding chapters address our basic approach towards the performance evaluation of parallel supercomputers targeted for use in numerical aerodynamic simulation.

Keywords: supercomputers, parallel computers, computational fluid dynamics, benchmarking, performance evaluation.

1. INTRODUCTION

Computational Fluid Dynamics (CFD) is one of the fields in the area of scientific computing that has driven the development of modern vector supercomputers. Availability of these high performance computers has led to impressive advancements in the state of the art of CFD, both in terms of the physical complexity of the simulated problems and the development of computational algorithms capable of extracting high levels of sustained performance. However, to carry out the computational simulations of future importance to the aerospace community, CFD must be able and ready to exploit potential performance and cost/performance gains possible through the use of highly parallel processing technologies. Use of parallel supercomputers appears to be one of the most promising avenues for realizing large complex physical simulations within realistic time and cost constraints. Although many of the current CFD application programs are amenable to a high degree of parallel computation, performance data on such codes for the current generation of parallel computers often has been less than remarkable. This is especially true for the class of CFD algorithms involving global data dependencies, commonly referred to as the implicit methods. Often the bottleneck is data motion, due to high latencies and inadequate bandwidth.

It is a common practice among computer hardware designers to use the dense linear equation solution subroutine in the LINPACK to represent the scientific computing workload. Unfortunately,

the computational structures in most CFD algorithms bear little resemblance to this LINPACK routine, both in terms of its parallelization strategy as well as floating point and memory reference features. Most CFD application codes are characterized by their use of either regular or irregular sparse data structures and associated algorithms. One of the reasons for this state of affairs is the near absence of communication between computer scientists engaged in the design of high performance parallel computers and the computational scientists involved in the development of CFD applications. In order to have a beneficial effect on the end product, such exchange of information should occur during the early stages of the design process. It appears that one of the contributing factors for this lack of effective communication is the complexity and confidentiality associated with the state-of-the-art CFD application codes. One way to help the design process is to provide the computer scientists with synthetic CFD application programs, which lack the complexity of a real application, but at the same time retain all the essential computational structures. Such synthetic application codes can be accompanied by detailed and simpler descriptions of the algorithms involved. In return, the performance data on such synthetic application codes can be used to evaluate different parallel supercomputer systems at the procurement stage by the CFD community.

Computational Fluid Dynamics involves the numerical solution of a system of nonlinear partial differential equations in two or three spatial dimensions, with or without time dependence. The governing partial differential equations, referred to as the Navier-Stokes equations, represent the laws of conservation of mass, momentum and energy applied to a fluid medium in motion. These equations, when supplemented by appropriate boundary and initial conditions describe a particular physical problem. To obtain a system of equations amenable to solution on a computer requires the discretization of the differential equations through the use of finite difference, finite volume, finite element or spectral methods. The inherent nonlinearities of the governing equations necessitate the use of iterative solution techniques. Over the past years, a variety of efficient numerical algorithms have been developed, all requiring many floating point operations and large amounts of computer memory to achieve a solution with a desired level of accuracy.

In current CFD applications, there are two types of computational meshes used for the spatial discretization process: structured and unstructured. Structured meshes are characterized by a consistent, logical ordering of mesh points, whose connectivity is associated with a rectilinear coordinate system. Computationally, structured meshws give rise to regularly strided memory reference characteristics. In contrast, unstructured meshes offer greater freedom in terms of mesh point distribution, but require the generation and storage of random connectivity information. Computationally, this results in indirect memory addressing with random strides, with its attendant increase in memory bandwidth requirements. The synthetic application codes currently under consideration are restricted to the case of structured meshes.

The numerical solution algorithms used in CFD codes can be broadly categorized as either

explicit or implicit, based on the procedure used for the time domain integration. Among the advantages of the explicit schemes are the high degree of easily exploitable parallelism and the localized spatial data dependencies. These properties have resulted in highly efficient implementations of explicit CFD algorithms on a variety of current generation highly parallel processors. However, the explicit schemes suffer from stringent numerical stability bounds and as a result are not optimal for problems that require fine mesh spacing for numerical resolution. In contrast, implicit schemes have less stringent stability bounds and are suitable for problems involving highly stretched meshes. However, their parallel implementation is more difficult and involve local as well as global spatial data dependencies. In addition, some of the implicit algorithms possess limited degrees of exploitable parallelism. At present, we restrict our synthetic applications to three different representative implicit schemes found in a wide spectrum of production CFD codes in use at the NASA Ames Research center.

In the remaining sections of this chapter, we describe the development of a collection of synthetic application programs. First we discuss the rationale behind this approach followed by a complete description of three such synthetic applications. We also outline the problem setup along with the associated verification tests, when they are used to benchmark highly parallel systems.

2. RATIONALE

In the past, vector supercomputer performance was evaluated through the use of suites of kernels chosen to characterize generic computational structures present at a site's workload. For example, NAS Kernels ([1]) were selected to characterize the computational workloads inherent in a majority of algorithms used by the CFD community at the NASA Ames Research Center. However, for highly parallel computer systems, this approach is inadequate, for reasons outlined below.

The first stage of the pseudo application development process was the analysis of a variety of implicit CFD codes and the identification of a set of generic computational structures that represented a range of computational tasks embedded in them. As a result, the following computational kernels were selected:

a) Solution of multiple, independent systems of non diagonally-dominant, block tridiagonal equations with a (5×5) block size.

b) Solution of multiple, independent systems of non diagonally-dominant, scalar pentadiagonal equations.

c) Regular-sparse, block (5×5) matrix-vector multiplication.

d) Regular-sparse, block (5×5) lower and upper triangular system solution.

These kernels constitute a majority of the computationally-intensive, main building blocks of the CFD programs designed for the numerical solution of three-dimensional (3D), Euler/Navier-

Stokes equations using finite-volume/finite-difference discretization on structured grids. Kernels (a) and (b) are representative of the computations associated with the implicit operator in versions of the ARC3D code ([2]). These kernels involve global data dependencies. Although they are similar in many respects, there is a fundamental difference with regard to the communication-to-computation ratio. Kernel (c) typifies the computation of the explicit part of almost all CFD algorithms for structured grids. Here all data dependencies are local, with either nearest neighbor or at most next-to-nearest neighbor type dependencies. Kernel (d) represents the computations associated with the implicit operator of a newer class of implicit CFD algorithms, typified by the code INS3D-LU ([3]). This kernel may contain only a limited degree of parallelism, relative to the other kernels.

In terms of their parallel implementation, these kernels represent varying characteristics with regard to the following aspects, which are often related:

1) Available degree of parallelism.
2) Level of parallelism and granularity.
3) Data space partitioning strategies.
4) Global vs. local data dependencies.
5) Inter-processor and in-processor data motion requirements.
6) Ratio of communication-to-computation.

Previous research efforts in adapting algorithms in a variety of flow solvers to current generation of highly parallel processors have indicated that the overall performance of many CFD codes is critically dependent on the latency and bandwidth of both the in-processor and inter-processor data motion. Therefore, it is important for the integrity of the benchmarking process to faithfully reproduce a majority of the data motions encountered during the execution of applications in which these kernels are embedded. Also, the nature and amount of data motion is dependent on the kernel algorithms along with the associated data structures and the interaction of these kernels among themselves as well as with the remainder of the application that is outside their scope.

To obtain realistic performance data, specification of both the incoming and outgoing data structures of the kernels should mimic those occuring in an application program. The incoming data structure is dependent on the section of the code where the data is generated, not on the kernel. The optimum data structure for the kernel may turn out to be sub-optimal for the code segments where the data is generated and vice-versa. Similar considerations also apply to the outgoing data structure. Allowing the freedom to choose optimal incoming and outgoing data structures for the kernel as a basis for evaluating its performance is liable to produce results that are not applicable to a complete application code. The overall performance needs to reflect the cost of data motion that occur between kernels.

In order to reproduce most of the data motions encountered in the execution of these kernels in a typical CFD application, we propose embedding them in a pseudo application code. It is designed for the numerical solution of a synthetic system of non-linear Partial Differential Equations (PDE's), using iterative techniques similar to those found in CFD applications of interest to the NASA Ames Research Center. However, it contains none of the pre- and post-processing required by the full CFD applications, or the interactions of the processors and the I/O subsystem. This can be regarded as a stripped-down version of a CFD application. It retains the basic kernels that are the principal building blocks of the application and admits a majority of the interactions required between these basic routines. Also, the stripped-down version does not represent a fully configured CFD application in terms of system memory requirements. This fact has the potential for creating data partitioning strategies during the parallel implementation of the synthetic problem that may be inappopriate for the full application.

From a functionality point of view, the stripped-down version does not contain the algorithms used to apply boundary conditions as in a real application. It is well known that often, the boundary algorithms gives rise to load imbalances and idling of processors in highly parallel systems. Due to relaxing of this requirement, it is likely that the overall system performance and efficiency data obtained using the stripped-down version may be higher than that of an actual application. This effect is somewhat mitigated by the fact that for most realistic problems, only a relatively small time is spent dealing with boundary algorithms when compared to the time spent in dealing with the internal mesh points. Also, most boundary algorithms involve only local data dependencies.

Among the other advantages of the stripped-down application vs. full application approach are:

1) Allows benchmarking where real application codes are confidential.

2) Easier to manipulate and port from one system to another.

3) Since only the abstract algorithm is specified, it facilitates new implementations that are tied closely to the architecture under consideration.

4) Allows easy addition of other existing and emerging CFD algorithms to the benchmarking process.

5) Easily scalable to larger problem sizes.

It should be noted that this synthetic problem differs from a real CFD problem in the following important aspects:

1) In full CFD application codes, a non-orthogonal coordinate transformation ([2]) is used to map the complex physical domains to the regular computational domains, thereby introducing metric coefficients of the transformation into the governing PDE's and boundary conditions.

Such transformations are absent in the synthetic problem, and as a result may have a reduced arithmetic complexity and storage requirements.

2) A blend of nonlinear, second- and fourth-difference artificial dissipation terms ([4]) is used in most of the actual CFD codes, whose coefficients are determined based on the local changes in pressure. In the stripped-down version, only a linear, fourth difference term is used. This reduces the arithmetic and communication complexity needed to compute the added higher-order dissipation terms. However, it should be noted that computation of these artificial dissipation terms involve only local data dependencies, similar to the matrix-vector multiplication kernel.

3) In codes where artificial dissipation is not used, upwind differencing based on either flux-vector splitting ([5],[6]) or flux-difference splitting ([7]) or Total Variation Diminishing (TVD) schemes ([8]) is used. The absence of such differencing schemes in the stripped-down version induces effects similar to (2) on the performance data.

4) Absence of turbulence models. Computation of terms representing some turbulence models involve a combination of local and some long-range data dependencies. Arithmetic and communication complexity associated with turbulence models are absent.

In addition, it also needs to be emphasized that the stripped-down problem is neither designed nor is suitable for the purposes of evaluating the convergence rates and/or the applicability of various iterative linear system solvers used in computational fluid dynamics applications. As mentioned before, the synthetic problem differs from the real CFD applications in following important ways:

1) Absence of realistic boundary algorithms.

2) Higher than normal dissipative effects.

3) Lack of upwind differencing effects, based on either flux-vector splitting or TVD schemes.

4) Absence of geometric stiffness introduced through boundary conforming coordinate transformations and highly stretched meshes.

5) Lack of evolution of weak (i.e.,C^0-) solutions found in real CFD applications, during the iterative process.

6) Absence of turbulence modelling effects.

Some of these effects tend to suppress the predominantly hyperbolic nature exhibited by the Navier-Stokes equations, when describing compressible flows at high Reynolds numbers.

3. MATHEMATICAL PROBLEM DEFINITION

We consider the numerical solution of the following synthetic system of five nonlinear partial differential equations (PDE's):

$$\frac{\partial \mathbf{U}}{\partial \tau} = \frac{\partial \mathbf{E}(\mathbf{U})}{\partial \xi} + \frac{\partial \mathbf{F}(\mathbf{U})}{\partial \eta} + \frac{\partial \mathbf{G}(\mathbf{U})}{\partial \zeta} + \frac{\partial \mathbf{T}(\mathbf{U}, \mathbf{U}_\xi)}{\partial \xi} + \frac{\partial \mathbf{V}(\mathbf{U}, \mathbf{U}_\eta)}{\partial \eta} + \frac{\partial \mathbf{W}(\mathbf{U}, \mathbf{U}_\zeta)}{\partial \zeta} + \mathbf{H}(\mathbf{U}, \mathbf{U}_\xi, \mathbf{U}_\eta, \mathbf{U}_\zeta), \qquad (\tau, \xi, \eta, \zeta) \in D_\tau \times D \tag{3.1a}$$

with the boundary conditions:

$$\mathcal{B}(\mathbf{U}, \mathbf{U}_\xi, \mathbf{U}_\eta, \mathbf{U}_\zeta) = \mathbf{U}^{\mathbf{B}}(\tau, \xi, \eta, \zeta), \qquad (\tau, \xi, \eta, \zeta) \in \mathcal{D}_\tau \times \partial\mathcal{D} \tag{3.1b}$$

and initial conditions:

$$\mathbf{U} = \mathbf{U}^0(\xi, \eta, \zeta), \qquad (\xi, \eta, \zeta) \in D \quad \text{for} \quad \tau = 0, \tag{3.1c}$$

where $D \in \Re^3$ is a bounded domain, ∂D is its boundary and $D_\tau = \{0 \leq \tau \leq T\}$.

Also, the solution to the system of PDE's:

$$\mathbf{U} = \begin{pmatrix} u^{(1)} \\ u^{(2)} \\ u^{(3)} \\ u^{(4)} \\ u^{(5)} \end{pmatrix} \tag{3.2}$$

defined in $(D \cup \partial D) \times D_\tau$, is a vector function of temporal variable τ and spatial variables (ξ, η, ζ) that form the orthogonal coordinate system in $\Re^3$, i.e.;

$$u^{(m)} = u^{(m)}(\tau, \xi, \eta, \zeta).$$

The vector functions $\mathbf{U}^B$ and $\mathbf{U}^0$ are given and $\mathcal{B}$ is the boundary operator. **E**, **F**, **G**, **T**, **V**, **W** and **H** are vector functions with five components each of the form:

$$\mathbf{E} = \begin{pmatrix} e^{(1)} \\ e^{(2)} \\ e^{(3)} \\ e^{(4)} \\ e^{(5)} \end{pmatrix} \tag{3.3}$$

and $e^{(m)} = e^{(m)}(\mathbf{U})$ etc. are prescribed functions.

The system given by Eq.(3.1a) is in the 'normal-form', i.e., it gives explicitly the time derivatives of all the dependent variables $u^{(1)}, u^{(2)}, \ldots, u^{(5)}$. Consequently, the Cauchy data at $\tau = 0$, given by Eq.(3.1c) permits the calculation of solution $\mathbf{U}(\tau, \xi, \eta, \zeta)$ for $\tau > 0$.

In the current implementation of the synthetic PDE system solver, we seek a steady-state solution of Eq.(3.1) of the form:

$$\mathbf{U}^* = \mathbf{f}(\xi, \eta, \zeta) = \begin{pmatrix} f^{(1)} \\ f^{(2)} \\ f^{(3)} \\ f^{(4)} \\ f^{(5)} \end{pmatrix} \tag{3.4}$$

where $f^{(m)} = f^{(m)}(\xi, \eta, \zeta)$ are prescribed functions of the following form:

$$\begin{pmatrix} f^{(1)}(\xi,\eta,\zeta) \\ f^{(2)}(\xi,\eta,\zeta) \\ f^{(3)}(\xi,\eta,\zeta) \\ f^{(4)}(\xi,\eta,\zeta) \\ f^{(5)}(\xi,\eta,\zeta) \end{pmatrix} = \begin{pmatrix} C_{1,1} & C_{1,2} & \cdots & C_{1,13} \\ C_{2,1} & C_{2,2} & \cdots & C_{2,13} \\ \vdots & \vdots & \ddots & \vdots \\ C_{5,1} & C_{5,2} & \cdots & C_{5,13} \end{pmatrix} \mathbf{e}(\xi, \eta, \zeta) \tag{3.5}$$

Here, the vector $\mathbf{e}$ is given by:

$$\mathbf{e}^T = (\,1 \quad \xi \quad \eta \quad \zeta \quad \xi^2 \quad \eta^2 \quad \zeta^2 \quad \xi^3 \quad \eta^3 \quad \zeta^3 \quad \xi^4 \quad \eta^4 \quad \zeta^4\,),$$

and $C_{m,n}, m = 1, 2, \ldots, 5$, $n = 1, 2, \ldots, 13$ are specified constants.

The vector forcing function $\mathbf{H} = [h^{(1)}, h^{(2)}, h^{(3)}, h^{(4)}, h^{(5)}]^T$, where $h^{(m)} = h^{(m)}(\xi, \eta, \zeta)$ is chosen such that the system of PDE's, along with its boundary and initial conditions, satisfies the prescribed exact solution, $\mathbf{U}^*$. This implies:

$$\begin{aligned} \mathbf{H}^*(\xi,\eta,\zeta) = -[&\frac{\partial \mathbf{E}(\mathbf{U}^*)}{\partial \xi} + \frac{\partial \mathbf{F}(\mathbf{U}^*)}{\partial \eta} + \frac{\partial \mathbf{G}(\mathbf{U}^*)}{\partial \zeta} \\ &+ \frac{\partial \mathbf{T}(\mathbf{U}^*, \mathbf{U}^*_\xi)}{\partial \xi} + \frac{\partial \mathbf{V}(\mathbf{U}^*, \mathbf{U}^*_\eta)}{\partial \eta} + \frac{\partial \mathbf{W}(\mathbf{U}^*, \mathbf{U}^*_\zeta)}{\partial \zeta}], \qquad \text{for } (\xi,\eta,\zeta) \in \mathrm{D} \times \mathrm{D}_\tau \end{aligned} \tag{3.6}$$

The bounded spatial domain D is specified to be the interior of the unit cube $[(0,1) \times (0,1) \times (0,1)]$, i.e.:

$$D = \{(\xi, \eta, \zeta) : 0 < \xi < 1, 0 < \eta < 1, 0 < \zeta < 1\}$$

and its boundary ∂D, is the surface of the unit cube given by:

$$\partial D = \{(\xi,\eta,\zeta) : \xi = 0 \text{ or } 1\} \cup \{(\xi,\eta,\zeta) : \eta = 0 \text{ or } 1\} \cup \{(\xi,\eta,\zeta) : \zeta = 0 \text{ or } 1\}$$

The vector functions $\mathbf{E}, \mathbf{F}, \mathbf{G}, \mathbf{T}, \mathbf{V}$ and $\mathbf{W}$ of the synthetic problem are specified to be the following:

$$\mathbf{E} = \begin{pmatrix} -u^{(2)} \\ -[u^{(2)}]^2/u^{(1)} - \phi \\ -[u^{(2)}u^{(3)}]/u^{(1)} \\ -[u^{(2)}u^{(4)}]/u^{(1)} \\ -[u^{(2)}/u^{(1)}][u^{(5)} + \phi] \end{pmatrix}; \qquad \mathbf{F} = \begin{pmatrix} -u^{(3)} \\ -[u^{(2)}u^{(3)}]/u^{(1)} \\ -[u^{(3)}]^2/u^{(1)} - \phi \\ -[u^{(3)}u^{(4)}]/u^{(1)} \\ -[u^{(3)}/u^{(1)}][u^{(5)} + \phi] \end{pmatrix}$$

$$\mathbf{G} = \begin{pmatrix} -u^{(4)} \\ -[u^{(2)}u^{(4)}]/u^{(1)} \\ -[u^{(3)}u^{(4)}]/u^{(1)} \\ -[u^{(4)}]^2/u^{(1)} - \phi \\ -[u^{(4)}/u^{(1)}][u^{(5)} + \phi] \end{pmatrix}$$

where,

$$\phi = k_2\{u^{(5)} - 0.5[\frac{([u^{(2)}]^2 + [u^{(3)}]^2 + [u^{(4)}]^2)}{u^{(1)}}]\}.$$

Also,

$$\mathbf{T} = \begin{pmatrix} d_\xi^{(1)}(\partial u^{(1)}/\partial\xi) \\ d_\xi^{(2)}(\partial u^{(2)}/\partial\xi) + (4./3.)k_3k_4(\partial[u^{(2)}/u^{(1)}]/\partial\xi) \\ d_\xi^{(3)}(\partial u^{(3)}/\partial\xi) + k_3k_4(\partial[u^{(3)}/u^{(1)}]/\partial\xi) \\ d_\xi^{(4)}(\partial u^{(4)}/\partial\xi) + k_3k_4(\partial[u^{(4)}/u^{(1)}]/\partial\xi) \\ t^{(5)} \end{pmatrix},$$

where,

$$t^{(5)} = d_\xi^{(5)}\frac{\partial u^{(5)}}{\partial\xi}$$
$$+ 0.5(1. - k_1k_5)\frac{\partial}{\partial\xi}(\frac{[u^{(2)}]^2 + [u^{(3)}]^2 + [u^{(4)}]^2}{[u^{(1)}]^2}) + (\frac{1.}{6.})\frac{\partial}{\partial\xi}[u^{(2)}/u^{(1)}]^2 + k_1k_5\frac{\partial}{\partial\xi}[u^{(5)}/u^{(1)}].$$

$$\mathbf{V} = \begin{pmatrix} d_\eta^{(1)}(\partial u^{(1)}/\partial\eta) \\ d_\eta^{(2)}(\partial u^{(2)}/\partial\eta) + k_3k_4(\partial[u^{(2)}/u^{(1)}]/\partial\eta) \\ d_\eta^{(3)}(\partial u^{(3)}/\partial\eta) + (4./3.)k_3k_4(\partial[u^{(3)}/u^{(1)}]/\partial\eta) \\ d_\eta^{(4)}(\partial u^{(4)}/\partial\eta) + k_3k_4(\partial[u^{(4)}/u^{(1)}]/\partial\eta) \\ v^{(5)} \end{pmatrix},$$

where,

$$v^{(5)} = d_\eta^{(5)}\frac{\partial u^{(5)}}{\partial\eta} + 0.5(1. - k_1k_5)\frac{\partial}{\partial\eta}\left(\frac{[u^{(2)}]^2 + [u^{(3)}]^2 + [u^{(4)}]^2}{[u^{(1)}]^2}\right) + \left(\frac{1.}{6.}\right)\frac{\partial}{\partial\eta}[u^{(3)}/u^{(1)}]^2 + k_1k_5\frac{\partial}{\partial\eta}[u^{(5)}/u^{(1)}].$$

$$\mathbf{W} = \begin{pmatrix} d_\zeta^{(1)}(\partial u^{(1)}/\partial\zeta) \\ d_\zeta^{(2)}(\partial u^{(2)}/\partial\zeta) + k_3k_4(\partial[u^{(2)}/u^{(1)}]/\partial\zeta) \\ d_\zeta^{(3)}(\partial u^{(3)}/\partial\zeta) + k_3k_4(\partial[u^{(3)}/u^{(1)}]/\partial\zeta) \\ d_\zeta^{(4)}(\partial u^{(4)}/\partial\zeta) + (4./3.)k_3k_4(\partial[u^{(4)}/u^{(1)}]/\partial\zeta) \\ w^{(5)} \end{pmatrix},$$

where,

$$w^{(5)} = d_\zeta^{(5)}\frac{\partial u^{(5)}}{\partial\zeta} + 0.5(1. - k_1k_5)\frac{\partial}{\partial\zeta}\left(\frac{[u^{(2)}]^2 + [u^{(3)}]^2 + [u^{(4)}]^2}{[u^{(1)}]^2}\right) + \left(\frac{1.}{6.}\right)\frac{\partial}{\partial\zeta}[u^{(4)}/u^{(1)}]^2 + k_1k_5\frac{\partial}{\partial\zeta}[u^{(5)}/u^{(1)}],$$

and k_1, k_2, k_3, k_4, k_5, $d_\xi^{(m)}$, $d_\eta^{(m)}$, $d_\zeta^{(m)}$ $(m = 1, 2, \ldots, 5)$ are given constatnts.

3.1 THE BOUNDARY CONDITIONS

The boundary conditions for the system of PDE's is prescribed to be of the uncoupled Dirichlet type, and is specified to be compatible with $\mathbf{U}^*$, such that:

$$u^{(m)} = f^{(m)}(\xi, \eta, \zeta), \qquad \text{for} \quad (\tau, \xi, \eta, \zeta) \in D_\tau \times \partial D \tag{3.7}$$

and $m = 1, 2, \ldots, 5$.

3.2. THE INITIAL CONDITIONS

The initial values $\mathbf{U}^0$ in D are set to those obtained by a transfinite, tri-linear interpolation ([9]), of the boundary data given by Eq.(3.7). Let:

$$\begin{aligned} P_\xi^{(m)} &= (1.-\xi)\, u^{(m)}(0,\eta,\zeta) + \xi\, u^{(m)}(1,\eta,\zeta), \\ P_\eta^{(m)} &= (1.-\eta)\, u^{(m)}(\xi,0,\zeta) + \eta\, u^{(m)}(\xi,1,\zeta), \\ P_\zeta^{(m)} &= (1.-\zeta)\, u^{(m)}(\xi,\eta,0) + \zeta\, u^{(m)}(\xi,\eta,1). \end{aligned} \tag{3.8}$$

Then,

$$\begin{aligned} u^{(m)}(\tau=0,\xi,\eta,\zeta) = P_\xi^{(m)} + P_\eta^{(m)} + P_\zeta^{(m)} \\ - P_\xi^{(m)} P_\eta^{(m)} - P_\eta^{(m)} P_\zeta^{(m)} - P_\zeta^{(m)} P_\xi^{(m)} \\ + P_\xi^{(m)} P_\eta^{(m)} P_\zeta^{(m)}, \qquad \text{for } (\xi,\eta,\zeta) \in \mathrm{D} \end{aligned} \tag{3.9}$$

4. THE NUMERICAL SCHEME

Starting from the initial values prescribed by Eq.(3.9), we seek some discrete approximation $\mathbf{U}_h^* \in D$ to the steady-state solution $\mathbf{U}^*$ of Eq.(3.1), through the numerical solution of the nonlinear system of PDE's using a pseudo-time marching scheme and a spatial discretization procedure based on finite difference approximations.

4.1. IMPLICIT TIME DIFFERENCING

The independent temporal variable τ is discretized to produce the set:

$$D_\tau = \{\tau_n : n \in [0, N]\}$$

where the discrete time increment $\Delta\tau$ is given by:

$$\tau_n = \tau_{n-1} + \Delta\tau = n\Delta\tau. \tag{4.1}$$

Also the discrete approximation of $\mathbf{U}$ on D_τ is denoted by:

$$\mathbf{U}(\tau) \approx \mathbf{U}^\tau(n\Delta\tau) = \mathbf{U}^n. \tag{4.2}$$

A generalized single-step temporal differencing scheme for advancing the solution of Eq.(3.1) is given by ([10]):

$$\Delta\mathbf{U}^n = \frac{\beta\Delta\tau}{(1+\theta)}\frac{\partial\Delta\mathbf{U}^n}{\partial\tau} + \frac{\Delta\tau}{(1+\theta)}\frac{\partial\mathbf{U}^n}{\partial\tau} + \frac{\theta}{(1+\theta)}\Delta\mathbf{U}^{n-1} + O[(\beta - \frac{1}{2} - \theta)\Delta\tau^2 + \Delta\tau^3]. \tag{4.3}$$

where the forward difference operator Δ is defined as:

$$\Delta \mathbf{U}^n = \mathbf{U}^{n+1} - \mathbf{U}^n. \tag{4.4}$$

With the appropriate choice for the parameters β and θ, Eq.(4.3) reproduces many of the well known two- and three-level, explicit and implicit schemes. In this particular case, we are interested only in the two-level, first-order accurate, Euler implicit scheme given by $\beta = 1$ and $\theta = 0$, i.e.:

$$\Delta \mathbf{U}^n = \Delta\tau \frac{\partial \Delta \mathbf{U}^n}{\partial \tau} + \Delta\tau \frac{\partial \mathbf{U}^n}{\partial \tau} + O[\Delta\tau^2]. \tag{4.5}$$

Substituting for $(\partial \Delta \mathbf{U}^n/\partial\tau)$ and $(\partial \mathbf{U}/\partial\tau)$ in Eq.(4.5), using Eq.(3.1a), we get;

$$\begin{aligned} \Delta \mathbf{U}^n = \Delta\tau[&\frac{\partial(\Delta \mathbf{E}^n + \Delta \mathbf{T}^n)}{\partial\xi} + \frac{\partial(\Delta \mathbf{F}^n + \Delta \mathbf{V}^n)}{\partial\eta} + \frac{\partial(\Delta \mathbf{G}^n + \Delta \mathbf{W}^n)}{\partial\zeta}] \\ &+ \Delta\tau[\frac{\partial(\mathbf{E}+\mathbf{T})^n}{\partial\xi} + \frac{\partial(\mathbf{F}+\mathbf{V})^n}{\partial\eta} + \frac{\partial(\mathbf{G}+\mathbf{W})^n}{\partial\zeta}] + \Delta\tau \mathbf{H}^*. \end{aligned} \tag{4.6}$$

where $\Delta \mathbf{E}^n = \mathbf{E}^{n+1} - \mathbf{E}^n$ and $\mathbf{E}^{n+1} = \mathbf{E}(\mathbf{U}^{n+1})$ etc.

Eq.(4.6) is nonlinear in $\Delta \mathbf{U}^n$ as a consequence of the fact that the increments $\Delta \mathbf{E}^n$, $\Delta \mathbf{F}^n$, $\Delta \mathbf{G}^n$, $\Delta \mathbf{T}^n$, $\Delta \mathbf{V}^n$ and $\Delta \mathbf{W}^n$ are nonlinear functions of the dependent variables $\mathbf{U}$ and its derivatives $\mathbf{U}_\xi$, $\mathbf{U}_\eta$ and $\mathbf{U}_\zeta$. A linear equation with the same temporal accuracy as Eq.(4.6) can be obtained by a linearization procedure using a local Taylor series expansion in time about $\mathbf{U}^n$, ([11],[12]);

$$\begin{aligned} \mathbf{E}^{n+1} &= \mathbf{E}^n + (\frac{\partial \mathbf{E}}{\partial\tau})^n \Delta\tau + O(\Delta\tau^2) \\ &= \mathbf{E}^n + (\frac{\partial \mathbf{E}}{\partial \mathbf{U}})^n (\frac{\partial \mathbf{U}}{\partial\tau})^n \Delta\tau + O(\Delta\tau^2). \end{aligned} \tag{4.7}$$

Also,

$$\mathbf{U}^{n+1} = \mathbf{U}^n + (\frac{\partial \mathbf{U}}{\partial\tau})^n \Delta\tau + O(\Delta\tau^2). \tag{4.8}$$

Then, by combining Eq.(4.7) and (4.8), we get;

$$\mathbf{E}^{n+1} = \mathbf{E}^n + (\frac{\partial \mathbf{E}}{\partial \mathbf{U}})^n (\mathbf{U}^{n+1} - \mathbf{U}^n) + O(\Delta\tau^2). \tag{4.9}$$

or,

$$\Delta \mathbf{E}^n = \mathbf{A}^n(\mathbf{U}) \Delta \mathbf{U}^n + O(\Delta\tau^2). \tag{4.10}$$

where $\mathbf{A}(\mathbf{U})$ is the Jacobian matrix $(\partial \mathbf{E}/\partial \mathbf{U})$.

It should be noted that the above non-iterative, time-linearization formulation does not lower the formal order of accuracy of temporal discretization in Eq.(4.6). However, if the steady-state

solution is the only objective, the quadratic convergence of the Newton-Raphson method (for sufficiently good inital approximations) is recovered only as $\Delta\tau \to \infty$.

Similarly, the linearization of remaining terms gives;

$$\begin{aligned}\Delta\mathbf{F}^n &= (\frac{\partial\mathbf{F}}{\partial\mathbf{U}})^n\Delta\mathbf{U}^n + O(\Delta\tau^2)\\ &= \mathbf{B}^n\Delta\mathbf{U}^n + O(\Delta\tau^2).\end{aligned} \tag{4.11a}$$

$$\begin{aligned}\Delta\mathbf{G}^n &= (\frac{\partial\mathbf{G}}{\partial\mathbf{U}})^n\Delta\mathbf{U}^n + O(\Delta\tau^2)\\ &= \mathbf{C}^n\Delta\mathbf{U}^n + O(\Delta\tau^2).\end{aligned} \tag{4.11b}$$

$$\begin{aligned}\Delta\mathbf{T}^n &= (\frac{\partial\mathbf{T}}{\partial\mathbf{U}})^n\Delta\mathbf{U}^n + (\frac{\partial\mathbf{T}}{\partial\mathbf{U}_\xi})^n\Delta\mathbf{U}_\xi^n + O(\Delta\tau^2)\\ &= \mathbf{M}^n\Delta\mathbf{U}^n + \mathbf{N}^n\Delta\mathbf{U}_\xi^n + O(\Delta\tau^2)\\ &= (\mathbf{M} - \mathbf{N}_\xi)^n\Delta\mathbf{U}^n + \frac{\partial(\mathbf{N}\Delta\mathbf{U})^n}{\partial\xi} + O(\Delta\tau^2).\end{aligned} \tag{4.11c}$$

$$\begin{aligned}\Delta\mathbf{V}^n &= (\frac{\partial\mathbf{V}}{\partial\mathbf{U}})^n\Delta\mathbf{U}^n + (\frac{\partial\mathbf{V}}{\partial\mathbf{U}_\eta})^n\Delta\mathbf{U}_\eta^n + O(\Delta\tau^2)\\ &= (\mathbf{P} - \mathbf{Q}_\eta)^n\Delta\mathbf{U}^n + \frac{\partial(\mathbf{Q}\Delta\mathbf{U})^n}{\partial\eta} + O(\Delta\tau^2).\end{aligned} \tag{4.11d}$$

$$\begin{aligned}\Delta\mathbf{W}^n &= (\frac{\partial\mathbf{W}}{\partial\mathbf{U}})^n\Delta\mathbf{U}^n + (\frac{\partial\mathbf{W}}{\partial\mathbf{U}_\zeta})^n\Delta\mathbf{U}_\zeta^n + O(\Delta\tau^2)\\ &= (\mathbf{R} - \mathbf{S}_\zeta)^n\Delta\mathbf{U}^n + \frac{\partial(\mathbf{S}\Delta\mathbf{U})^n}{\partial\zeta} + O(\Delta\tau^2).\end{aligned} \tag{4.11e}$$

where,

$$\mathbf{B}(\mathbf{U}) = \frac{\partial\mathbf{F}}{\partial\mathbf{U}}. \tag{4.12a}$$

$$\mathbf{C}(\mathbf{U}) = \frac{\partial\mathbf{G}}{\partial\mathbf{U}}. \tag{4.12b}$$

$$\mathbf{M}(\mathbf{U},\mathbf{U}_\xi) = \frac{\partial\mathbf{T}}{\partial\mathbf{U}}; \qquad \mathbf{N}(\mathbf{U}) = \frac{\partial\mathbf{T}}{\partial\mathbf{U}_\xi}; \qquad \mathbf{N}_\xi(\mathbf{U},\mathbf{U}_\xi) = \frac{\partial\mathbf{N}}{\partial\xi}. \tag{4.12c}$$

$$\mathbf{P}(\mathbf{U},\mathbf{U}_\eta) = \frac{\partial\mathbf{V}}{\partial\mathbf{U}}; \qquad \mathbf{Q}(\mathbf{U}) = \frac{\partial\mathbf{V}}{\partial\mathbf{U}_\eta}; \qquad \mathbf{Q}_\eta(\mathbf{U},\mathbf{U}_\eta) = \frac{\partial\mathbf{Q}}{\partial\eta}. \tag{4.12d}$$

$$\mathbf{R}(\mathbf{U},\mathbf{U}_\zeta) = \frac{\partial\mathbf{W}}{\partial\mathbf{U}}; \qquad \mathbf{S}(\mathbf{U}) = \frac{\partial\mathbf{W}}{\partial\mathbf{U}_\zeta}; \qquad \mathbf{S}_\zeta(\mathbf{U},\mathbf{U}_\zeta) = \frac{\partial\mathbf{S}}{\partial\zeta}. \tag{4.12e}$$

When the approximations given by Eq.(4.11) are introduced into Eq.(4.6), we obtain the following linear equation for $\Delta\mathbf{U}^n$;

$$\{\mathbf{I} - \Delta\tau[\frac{\partial(\mathbf{A}+\mathbf{M}-\mathbf{N}_\xi)^n}{\partial\xi} + \frac{\partial^2(\mathbf{N})^n}{\partial\xi^2} + \frac{\partial(\mathbf{B}+\mathbf{P}-\mathbf{Q}_\eta)^n}{\partial\eta} + \frac{\partial^2(\mathbf{Q})^n}{\partial\eta^2}$$
$$+ \frac{\partial(\mathbf{C}+\mathbf{R}-\mathbf{S}_\zeta)^n}{\partial\zeta} + \frac{\partial^2(\mathbf{S})^n}{\partial\zeta^2}]\}\Delta\mathbf{U}^n = \tag{4.13}$$
$$\Delta\tau[\frac{\partial(\mathbf{E}+\mathbf{T})^n}{\partial\xi} + \frac{\partial(\mathbf{F}+\mathbf{V})^n}{\partial\eta} + \frac{\partial(\mathbf{G}+\mathbf{W})^n}{\partial\zeta} + \mathbf{H}^*].$$

It should be noted that the notation of the form:

$$[\frac{\partial(\mathbf{A}+\mathbf{M}-\mathbf{N}_\xi)^n}{\partial\xi}]\Delta\mathbf{U}^n$$

is used to represent the expressions as such:

$$\frac{\partial[(\mathbf{A}+\mathbf{M}-\mathbf{N}_\xi)^n\Delta\mathbf{U}^n]}{\partial\xi}, \quad \text{etc.}$$

The left hand side (i.e.,**LHS**) of Eq.(4.13) is referred to as the implicit part and the right hand side (i.e.,**RHS**) as the explicit part.

The solution at the advanced time, $\tau = (n+1)\Delta\tau$ is given by:

$$\mathbf{U}^{n+1} = \mathbf{U}^n + \Delta\mathbf{U}^n. \tag{4.14}$$

The Jacobian matrices for the problem under consideration are given by:

$$\mathbf{A} = \begin{pmatrix} 0 & -1 & 0 & 0 & 0 \\ [u^{(2)}/u^{(1)}]^2 - q & (k_2-2)[u^{(2)}/u^{(1)}] & k_2[u^{(3)}/u^{(1)}] & k_2[u^{(4)}/u^{(1)}] & -k_2 \\ [u^{(2)}u^{(3)}]/[u^{(1)}]^2 & -[u^{(3)}/u^{(1)}] & -[u^{(2)}/u^{(1)}] & 0 & 0 \\ [u^{(2)}u^{(4)}]/[u^{(1)}]^2 & -[u^{(4)}/u^{(1)}] & 0 & -[u^{(2)}/u^{(1)}] & 0 \\ a_{51} & a_{52} & k_2[u^{(2)}u^{(3)}]/[u^{(1)}]^2 & k_2[u^{(2)}u^{(4)}]/[u^{(1)}]^2 & -k_1[u^{(2)}/u^{(1)}] \end{pmatrix}$$

where,

$$q = (\frac{k_2}{2})\{\frac{[u^{(2)}]^2 + [u^{(3)}]^2 + [u^{(4)}]^2}{[u^{(1)}]^2}\},$$

$$a_{51} = \{k_1[u^{(5)}/u^{(1)}] - 2q\}[\frac{u^{(2)}}{u^{(1)}}],$$

$$a_{52} = (\frac{k_2}{2})\{\frac{3[u^{(2)}]^2 + [u^{(3)}]^2 + [u^{(4)}]^2}{[u^{(1)}]^2}\} - k_1[\frac{u^{(5)}}{u^{(1)}}].$$

$$\mathbf{B} = \begin{pmatrix} 0 & 0 & -1 & 0 & 0 \\ [u^{(2)}u^{(3)}]/[u^{(1)}]^2 & -[u^{(3)}/u^{(1)}] & -[u^{(2)}/u^{(1)}] & 0 & 0 \\ [u^{(3)}/u^{(1)}]^2 - q & k_2[u^{(2)}/u^{(1)}] & (k_2-2)[u^{(3)}/u^{(1)}] & k_2[u^{(4)}/u^{(1)}] & -k_2 \\ [u^{(3)}u^{(4)}]/[u^{(1)}]^2 & 0 & -[u^{(4)}/u^{(1)}] & [u^{(3)}/u^{(1)}] & 0 \\ b_{51} & k_2[u^{(2)}u^{(3)}]/[u^{(1)}]^2 & b_{53} & k_2[u^{(3)}u^{(4)}]/[u^{(1)}]^2 & -k_1[u^{(3)}/u^{(1)}] \end{pmatrix}$$

where,

$$b_{51} = \{k_1[u^{(5)}/u^{(1)}] - 2q\}[\frac{u^{(3)}}{u^{(1)}}],$$

$$b_{53} = (\frac{k_2}{2})\{\frac{[u^{(2)}]^2 + 3[u^{(3)}]^2 + [u^{(4)}]^2}{[u^{(1)}]^2}\} - k_1[\frac{u^{(5)}}{u^{(1)}}].$$

$$\mathbf{C} = \begin{pmatrix} 0 & 0 & 0 & -1 & 0 \\ [u^{(2)}u^{(4)}]/[u^{(1)}]^2 & -[u^{(4)}/u^{(1)}] & 0 & -[u^{(2)}/u^{(1)}] & 0 \\ [u^{(3)}u^{(4)}]/[u^{(1)}]^2 & 0 & -[u^{(4)}/u^{(1)}] & [u^{(3)}/u^{(1)}] & 0 \\ [u^{(4)}/u^{(1)}]^2 - q & k_2[u^{(2)}/u^{(1)}] & k_2[u^{(3)}/u^{(1)}] & (k_2-2)[u^{(4)}/u^{(1)}] & -k_2 \\ c_{51} & k_2[u^{(2)}u^{(4)}]/[u^{(1)}]^2 & k_2[u^{(3)}u^{(4)}]/[u^{(1)}]^2 & c_{54} & -k_1[u^{(4)}/u^{(1)}] \end{pmatrix}$$

where,

$$c_{51} = \{k_1[\frac{u^{(5)}}{u^{(1)}}] - 2q\}[\frac{u^{(4)}}{u^{(1)}}],$$

$$c_{54} = (\frac{k_2}{2})\{\frac{[u^{(2)}]^2 + [u^{(3)}]^2 + 3[u^{(4)}]^2}{[u^{(1)}]^2}\} - k_1[\frac{u^{(5)}}{u^{(1)}}].$$

$$(\mathbf{M} - \mathbf{N}_\xi) = [\emptyset].$$

$$\mathbf{N} = \begin{pmatrix} d_\xi^{(1)} & 0 & 0 & 0 & 0 \\ -[4./3.]k_3k_4(u^{(2)}/[u^{(1)}]^2) & d_\xi^{(2)} + [4./3.]k_3k_4(1./u^{(1)}) & 0 & 0 & 0 \\ -k_3k_4(u^{(3)}/[u^{(1)}]^2) & 0 & d_\xi^{(3)} + k_3k_4(1./u^{(1)}) & 0 & 0 \\ -k_3k_4(u^{(4)}/[u^{(1)}]^2) & 0 & 0 & d_\xi^{(4)} + k_3k_4(1./u^{(1)}) & 0 \\ n_{51} & n_{52} & n_{53} & n_{54} & n_{55} \end{pmatrix},$$

where,

$$\begin{aligned} n_{51} =& -[(4./3.)k_3k_4 - k_1k_3k_4k_5]([u^{(2)}]^2/[u^{(1)}]^3) \\ & -[k_3k_4 - k_1k_3k_4k_5]([u^{(3)}]^2/[u^{(1)}]^3) \\ & -[k_3k_4 - k_1k_3k_4k_5]([u^{(4)}]^2/[u^{(1)}]^3) \\ & -k_1k_3k_4k_5(u^{(5)}/[u^{(1)}]^2), \\ n_{52} =& ([4./3.]k_3k_4 - k_1k_3k_4k_5)(u^{(2)}/[u^{(1)}]^2), \\ n_{53} =& (k_3k_4 - k_1k_3k_4k_5)(u^{(3)}/[u^{(1)}]^2), \\ n_{54} =& (k_3k_4 - k_1k_3k_4k_5)(u^{(4)}/[u^{(1)}]^2), \\ n_{55} =& d_\xi^{(5)} + (k_1k_3k_4k_5)(1./u^{(1)}). \end{aligned}$$

$$(\mathbf{P} - \mathbf{Q}_\eta) = [\emptyset].$$

$$\mathbf{Q} = \begin{pmatrix} d_\eta^{(1)} & 0 & 0 & 0 & 0 \\ -k_3k_4(u^{(2)}/[u^{(1)}]^2) & d_\eta^{(2)} + k_3k_4(1./u^{(1)}) & 0 & 0 & 0 \\ -(4./3.)k_3k_4(u^{(3)}/[u^{(1)}]^2) & 0 & d_\eta^{(3)} + (4./3.)k_3k_4(1./u^{(1)}) & 0 & 0 \\ -k_3k_4(u^{(4)}/[u^{(1)}]^2) & 0 & 0 & d_\eta^{(4)} + k_3k_4(1./u^{(1)}) & 0 \\ q_{51} & q_{52} & q_{53} & q_{54} & q_{55} \end{pmatrix},$$

where,

$$
\begin{aligned}
q_{51} =& -[k_3k_4 - k_1k_3k_4k_5]([u^{(2)}]^2/[u^{(1)}]^3) \\
& -[(4./3.)k_3k_4 - k_1k_3k_4k_5]([u^{(3)}]^2/[u^{(1)}]^3) \\
& -[k_3k_4 - k_1k_3k_4k_5]([u^{(4)}]^2/[u^{(1)}]^3) \\
& -k_1k_3k_4k_5(u^{(5)}/[u^{(1)}]^2), \\
q_{52} =& (k_3k_4 - k_1k_3k_4k_5)(u^{(2)}/[u^{(1)}]^2), \\
q_{53} =& ([4./3.]k_3k_4 - k_1k_3k_4k_5)(u^{(3)}/[u^{(1)}]^2), \\
q_{54} =& (k_3k_4 - k_1k_3k_4k_5)(u^{(4)}/[u^{(1)}]^2), \\
q_{55} =& d_\eta^{(5)} + (k_1k_3k_4k_5)(1./u^{(1)}).
\end{aligned}
$$

$$
(\mathbf{R} - \mathbf{S}_\zeta) = [\emptyset].
$$

$$
\mathbf{S} = \begin{pmatrix}
d_\zeta^{(1)} & 0 & 0 & 0 & 0 \\
-k_3k_4(u^{(2)}/[u^{(1)}]^2) & d_\zeta^{(2)} + k_3k_4(1./u^{(1)}) & 0 & 0 & 0 \\
-k_3k_4(u^{(3)}/[u^{(1)}]^2) & 0 & d_\zeta^{(3)} + k_3k_4(1./u^{(1)}) & 0 & 0 \\
-(4./3.)k_3k_4(u^{(4)}/[u^{(1)}]^2) & 0 & 0 & d_\zeta^{(4)} + (4./3.)k_3k_4(1./u^{(1)}) & 0 \\
s_{51} & s_{52} & s_{53} & s_{54} & s_{55}
\end{pmatrix},
$$

where,

$$
\begin{aligned}
s_{51} =& -[k_3k_4 - k_1k_3k_4k_5]([u^{(2)}]^2/[u^{(1)}]^3) \\
& -[k_3k_4 - k_1k_3k_4k_5]([u^{(3)}]^2/[u^{(1)}]^3) \\
& -[(4./3.)k_3k_4 - k_1k_3k_4k_5]([u^{(4)}]^2/[u^{(1)}]^3) \\
& -k_1k_3k_4k_5(u^{(5)}/[u^{(1)}]^2), \\
s_{52} =& (k_3k_4 - k_1k_3k_4k_5)(u^{(2)}/[u^{(1)}]^2), \\
s_{53} =& (k_3k_4 - k_1k_3k_4k_5)(u^{(3)}/[u^{(1)}]^2), \\
s_{54} =& ([4./3.]k_3k_4 - k_1k_3k_4k_5)(u^{(4)}/[u^{(1)}]^2), \\
s_{55} =& d_\zeta^{(5)} + (k_1k_3k_4k_5)(1./u^{(1)}).
\end{aligned}
$$

4.2. SPATIAL DISCRETIZATION

The independent spatial variables (ξ, η, ζ) are discretized by covering $\bar{D}$, (the closure of D), with a mesh of uniform increments (h_ξ, h_η, h_ζ) in each of the coordinate directions. The mesh points in the region will be identified by the index-triple (i, j, k), where the indices $i \in [1, N_\xi]$, $j \in [1, N_\eta]$ and $k \in [1, N_\zeta]$ correspond to the discretization of ξ, η and ζ coordinates respectively.

$$D_h \cup \partial D_h = \{(\xi_i, \eta_j, \zeta_k) : 1 \leq i \leq N_\xi, 1 \leq j \leq N_\eta, 1 \leq k \leq N_\zeta\}$$

where,

$$\xi_i = (i-1)h_\xi; \qquad \eta_j = (j-1)h_\eta; \qquad \zeta_k = (k-1)h_\zeta. \tag{4.15}$$

and the mesh widths are given by:

$$h_\xi = 1./(N_\xi - 1); \qquad h_\eta = 1./(N_\eta - 1); \qquad h_\zeta = 1./(N_\zeta - 1) \tag{4.16}$$

with $(N_\xi, N_\eta, N_\zeta) \in N$ being the number of mesh points in $\xi-$, $\eta-$ and $\zeta-$directions respectively.

Then, the set of interior mesh points is given by:

$$D_h = \{(\xi_i, \eta_j, \zeta_k) : 2 \leq i \leq (N_\xi - 1), 2 \leq j \leq (N_\eta - 1), 2 \leq k \leq (N_\zeta - 1)\}$$

and the boundary mesh points by:

$$\partial D_h = \{(\xi_i, \eta_j, \zeta_k) : i \in \{1, N_\xi\}\} \cup \{(\xi_i, \eta_j, \zeta_k) : j \in \{1, N_\eta\}\} \cup \{(\xi_i, \eta_j, \zeta_k) : k \in \{1, N_\zeta\}\}.$$

Also, the discrete approximation of U in $(\bar{D} \times D_\tau)$ is denoted by:

$$U(\tau, \xi, \eta, \zeta) \cong U_h^\tau(n\Delta\tau, (i-1)h_\xi, (j-1)h_\eta, (k-1)h_\zeta) = U_{i,j,k}^n. \tag{4.17}$$

4.3. SPATIAL DIFFERENCING

The spatial derivatives in Eq.(4.13) are approximated by the appropriate finite-difference quotients, based on the values of U_h^τ at mesh points in $D_h \cup \partial D_h$. We use three-point, second-order accurate central difference approximations in each of the three coordinate directions.

In the computation of the finite-difference approximation to the **RHS**, the following two general forms of spatial derivatives are encountered, i.e.:

$$\frac{\partial e^{(m)}(\mathbf{U})}{\partial \xi}$$

and

$$\frac{\partial t^{(m)}(\mathbf{U}, \mathbf{U}_\xi)}{\partial \xi}.$$

The first form is differenced as (using m-th component vector function **E** as an example):

$$\frac{\partial e^{(m)}(\mathbf{U})}{\partial \xi}|_{i,j,k} = (1/2h_\xi)[e^{(m)}(\mathbf{U}_{i+1,j,k}) - e^{(m)}(\mathbf{U}_{i-1,j,k})] + O(h_\xi^2). \tag{4.18}$$

The second form is approximated as (using m-th component of vector function **T** as an example):

$$\begin{aligned}\frac{\partial t^{(m)}(\mathbf{U},\mathbf{U}_\xi)}{\partial \xi}|_{i,j,k} = (1/h_\xi)\{t^{(m)}[(\frac{\mathbf{U}_{i+1,j,k}+\mathbf{U}_{i,j,k}}{2}),(\frac{\mathbf{U}_{i+1,j,k}-\mathbf{U}_{i,j,k}}{h_\xi})] \\ - t^{(m)}[(\frac{\mathbf{U}_{i,j,k}+\mathbf{U}_{i-1,j,k}}{2}),(\frac{\mathbf{U}_{i,j,k}-\mathbf{U}_{i-1,j,k}}{h_\xi})]\} + O(h_\xi^2),\end{aligned} \tag{4.19}$$

for $2 \le i \le (N_\xi - 1)$. Similar formulas are used in the $\eta-$ and $\zeta-$ directions as well.

During the finite-difference approximation of the spatial derivatives in the implicit part, following three general forms are encountered:

$$\frac{\partial[\mathbf{a}^{(m,l)}(\mathbf{U})\Delta u^{(l)}]}{\partial \xi},$$

$$\frac{\partial[\mathbf{m}^{(m,l)}(\mathbf{U},\mathbf{U}_\xi)\Delta u^{(l)}]}{\partial \xi},$$

and

$$\frac{\partial^2[\mathbf{n}^{(m,l)}(\mathbf{U})\Delta u^{(l)}]}{\partial \xi^2}.$$

The first form is approximated by the following:

$$\frac{\partial[\mathbf{a}^{(m,l)}(\mathbf{U})\Delta u^{(l)}]}{\partial \xi}|_{i,j,k} \approx [(1/2h_\xi)\{\mathbf{a}^{(m,l)}(\mathbf{U}_{i+1,j,k})\}]\Delta u^{(l)}_{i+1,j,k} - [(1/2h_\xi)\{\mathbf{a}^{(m,l)}(\mathbf{U}_{i-1,j,k})\}]\Delta u^{(l)}_{i-1,j,k}. \tag{4.20}$$

The second form is differenced in the following compact three-point form:

$$\begin{aligned}\frac{\partial[\mathbf{m}^{(m,l)}(\mathbf{U},\mathbf{U}_\xi)\Delta u^{(l)}]}{\partial \xi}|_{i,j,k} \approx [(1/2h_\xi)\{\mathbf{m}^{(m,l)}[(\frac{\mathbf{U}_{i+1,j,k}+\mathbf{U}_{i,j,k}}{2}),(\frac{\mathbf{U}_{i+1,j,k}-\mathbf{U}_{i,j,k}}{h_\xi})]\}]\Delta u^{(l)}_{i+1,j,k} \\ + [(1/2h_\xi)\{\mathbf{m}^{(m,l)}[(\frac{\mathbf{U}_{i+1,j,k}+\mathbf{U}_{i,j,k})}{2}),(\frac{\mathbf{U}_{i+1,j,k}-\mathbf{U}_{i,j,k}}{h_\xi})] \\ - \mathbf{m}^{(m,l)}[(\frac{\mathbf{U}_{i,j,k}+\mathbf{U}_{i-1,j,k}}{2}),(\frac{\mathbf{U}_{i,j,k}-\mathbf{U}_{i-1,j,k}}{h_\xi})]\}]\Delta u^{(l)}_{i,j,k} \\ - [(1/2h_\xi)\{\mathbf{m}^{(m,l)}[(\frac{\mathbf{U}_{i,j,k}+\mathbf{U}_{i-1,j,k}}{2}),(\frac{\mathbf{U}_{i,j,k}-\mathbf{U}_{i-1,j,k}}{h_\xi})]\}\Delta u^{(l)}_{i-1,j,k}.\end{aligned} \tag{4.21}$$

Finally, the third form is differenced as follows:

$$\begin{aligned}\frac{\partial^2[\mathbf{n}^{(m,l)}(\mathbf{U})\Delta u^{(l)}]}{\partial \xi^2}|_{i,j,k} \approx [(1/h_\xi^2)\{\mathbf{n}^{(m,l)}(\mathbf{U}_{i+1,j,k})\}]\Delta u^{(l)}_{i+1,j,k} \\ - [(2/h_\xi^2)\{\mathbf{n}^{(m,l)}(\mathbf{U}_{i,j,k})\}]\Delta u^{(l)}_{i,j,k} \\ + [(1/h_\xi^2)\{\mathbf{n}^{(m,l)}(\mathbf{U}_{i-1,j,k})\}]\Delta u^{(l)}_{i-1,j,k}.\end{aligned} \tag{4.22}$$

4.4. ADDED HIGHER ORDER DISSIPATION

When central difference type schemes are applied to the solution of the Euler and Navier-Stokes equations, a numerical dissipation model is included, and it plays an important role in determining their success. The form of the dissipation model is quite often a blending of second-difference and fourth-difference dissipation terms ([4]). The second-difference dissipation is nonlinear and is used to prevent oscillations in the neighborhood of discontinous (i.e.,C^0) solutions, and is negligible where the solution is smooth. The fourth-difference dissipation term is basically linear and is included to suppress high-frequency modes and allow the numerical scheme to converge to a steady state. Near solution discontinuities, it is reduced to zero. It is this term that affects the linear stability of the numerical scheme.

In the current implementation of the synthetic problem, a linear fourth- difference dissipation term of the following form:

$$-\Delta\tau\varepsilon\,[h_\xi^4\frac{\partial^4\mathbf{U}^n}{\partial\xi^4}+h_\eta^4\frac{\partial^4\mathbf{U}^n}{\partial\eta^4}+h_\zeta^4\frac{\partial^4\mathbf{U}^n}{\partial\zeta^4}],$$

is added to the right hand side of Eq.(3.1a). Here, ε is a specified constant. A similar term with $\mathbf{U}^n$ replaced by $\Delta\mathbf{U}^n$ will appear in the implicit operator, if it is desirable to treat the dissipation term implicitly as well.

In the interior of the computational domain, fourth-difference term is computed using the following five-point approximation:

$$h_\xi^4\frac{\partial^4\mathbf{U}^n}{\partial\xi^4}|_{i,j,k}\approx\mathbf{U}^n_{i+2,j,k}-4\mathbf{U}^n_{i+1,j,k}+6\mathbf{U}^n_{i,j,k}-4\mathbf{U}^n_{i-1,j,k}+\mathbf{U}^n_{i-2,j,k},\qquad(4.24a)$$

for $4\le i\le(N_\xi-3)$.

At the first two interior mesh points belonging to either end of the computational domain, the standard five point difference stencil used for the fourth-difference dissipation term is replaced by one-sided or one-sided biased stencils. These modifications are implemented so as to maintain a non-positive definite dissipation matrix for the system of difference equations ([16]).

Then, at $i=2$:

$$h_\xi^4\frac{\partial^4\mathbf{U}^n}{\partial\xi^4}|_{i,j,k}\approx\mathbf{U}^n_{i+2,j,k}-4\mathbf{U}^n_{i+1,j,k}+5\mathbf{U}^n_{i,j,k},\qquad(4.24b)$$

and at $i=3$:

$$h_\xi^4\frac{\partial^4\mathbf{U}^n}{\partial\xi^4}|_{i,j,k}\approx\mathbf{U}^n_{i+2,j,k}-4\mathbf{U}^n_{i+1,j,k}+6\mathbf{U}^n_{i,j,k}-4\mathbf{U}^n_{i-1,j,k}.\qquad(4.24c)$$

Also at $i=N_\xi-2$:

$$h_\xi^4\frac{\partial^4\mathbf{U}^n}{\partial\xi^4}|_{i,j,k}\approx-4\mathbf{U}^n_{i+1,j,k}+6\mathbf{U}^n_{i,j,k}-4\mathbf{U}^n_{i-1,j,k}+\mathbf{U}^n_{i-2,j,k},\qquad(4.24d)$$

and at $i = N_\xi - 1$:

$$h_\xi^4 \frac{\partial^4 \mathbf{U}^n}{\partial \xi^4}|_{i,j,k} \approx 5\mathbf{U}^n_{i,j,k} - 4\mathbf{U}^n_{i-1,j,k} + \mathbf{U}^n_{i-2,j,k}. \tag{4.24e}$$

Similar difference formulas are also used in the $\eta-$ and $\zeta-$ directions.

Also, for vector functions **T**, **V** and **W** used here:

$$(\mathbf{M} - \mathbf{N}_\xi) = 0, \qquad (\mathbf{P} - \mathbf{Q}_\eta) = 0, \qquad (\mathbf{R} - \mathbf{S}_\zeta) = 0.$$

Then, for the cases where added higher order dissipation terms are treated only explicitly, Eq.(4.13) reduces to:

$$\begin{aligned}\{\mathbf{I} - \Delta\tau[\frac{\partial(\mathbf{A})^n}{\partial\xi} + \frac{\partial^2(\mathbf{N})^n}{\partial\xi^2} + \frac{\partial(\mathbf{B})^n}{\partial\eta} + \frac{\partial^2(\mathbf{Q})^n}{\partial\eta^2} + \frac{\partial(\mathbf{C})^n}{\partial\zeta} + \frac{\partial^2(\mathbf{S})^n}{\partial\zeta^2}]\}\Delta\mathbf{U}^n = \\ \Delta\tau[\frac{\partial(\mathbf{E}+\mathbf{T})^n}{\partial\xi} + \frac{\partial(\mathbf{F}+\mathbf{V})^n}{\partial\eta} + \frac{\partial(\mathbf{G}+\mathbf{W})^n}{\partial\zeta}] \\ - \Delta\tau\,\varepsilon\,[h_\xi^4\frac{\partial^4\mathbf{U}^n}{\partial\xi^4} + h_\eta^4\frac{\partial^4\mathbf{U}^n}{\partial\eta^4} + h_\zeta^4\frac{\partial^4\mathbf{U}^n}{\partial\zeta^4}] \\ + \Delta\tau\mathbf{H}^*\end{aligned} \tag{4.25a}$$

When the implicit treatment is extended to the added higher order dissipation terms as well, Eq.(4.13), takes the following form:

$$\begin{aligned}\{\mathbf{I} - \Delta\tau[\ &\frac{\partial(\mathbf{A})^n}{\partial\xi} + \frac{\partial^2(\mathbf{N})^n}{\partial\xi^2} - \varepsilon h_\xi^4\frac{\partial^4(\mathbf{I})}{\partial\xi^4} \\ &+ \frac{\partial(\mathbf{B})^n}{\partial\eta} + \frac{\partial^2(\mathbf{Q})^n}{\partial\eta^2} - \varepsilon h_\eta^4\frac{\partial^4(\mathbf{I})}{\partial\eta^4} \\ &+ \frac{\partial(\mathbf{C})^n}{\partial\zeta} + \frac{\partial^2(\mathbf{S})^n}{\partial\zeta^2} - \varepsilon h_\zeta^4\frac{\partial^4(\mathbf{I})}{\partial\zeta^4}]\}\Delta\mathbf{U}^n = \\ &\Delta\tau[\frac{\partial(\mathbf{E}+\mathbf{T})^n}{\partial\xi} + \frac{\partial(\mathbf{F}+\mathbf{V})^n}{\partial\eta} + \frac{\partial(\mathbf{G}+\mathbf{W})^n}{\partial\zeta}] \\ &- \Delta\tau\varepsilon[h_\xi^4\frac{\partial^4\mathbf{U}^n}{\partial\xi^4} + h_\eta^4\frac{\partial^4\mathbf{U}^n}{\partial\eta^4} + h_\zeta^4\frac{\partial^4\mathbf{U}^n}{\partial\zeta^4}] \\ &+ \Delta\tau\mathbf{H}^*\end{aligned} \tag{4.25b}$$

The modified vector forcing function is given by:

$$\begin{aligned}\mathbf{H}^*(\xi,\eta,\zeta) = -[&\frac{\partial\mathbf{E}(\mathbf{U}^*)}{\partial\xi} + \frac{\partial\mathbf{F}(\mathbf{U}^*)}{\partial\eta} + \frac{\partial\mathbf{G}(\mathbf{U}^*)}{\partial\zeta} \\ &+ \frac{\partial\mathbf{T}(\mathbf{U}^*,\mathbf{U}^*_\xi)}{\partial\xi} + \frac{\partial\mathbf{V}(\mathbf{U}^*,\mathbf{U}^*_\eta)}{\partial\eta} + \frac{\partial\mathbf{W}(\mathbf{U}^*,\mathbf{U}^*_\zeta)}{\partial\zeta}] \\ &+ \varepsilon[h_\xi^4\frac{\partial^4\mathbf{U}^*}{\partial\xi^4} + h_\eta^4\frac{\partial^4\mathbf{U}^*}{\partial\eta^4} + h_\zeta^4\frac{\partial^4\mathbf{U}^*}{\partial\zeta^4}], \qquad \text{for } (\xi,\eta,\zeta) \in \mathrm{D}\times\mathrm{D}_\tau\end{aligned} \tag{4.26}$$

4.5. COMPUTATION OF THE EXPLICIT PART (i.e.,RHS)

The discretized form of the explicit part of Eq.(4.25) is given by:

$$
\begin{aligned}
[\mathbf{RHS}]^n|_{i,j,k} \approx \Delta\tau[D_\xi(\mathbf{E}+\mathbf{T})^n + D_\eta(\mathbf{F}+\mathbf{V})^n + D_\zeta(\mathbf{G}+\mathbf{W})^n]|_{i,j,k} \\
- \Delta\tau\,\varepsilon\,[h_\xi^4 D_\xi^4\mathbf{U}^n + h_\eta^4 D_\eta^4\mathbf{U}^n + h_\zeta^4 D_\zeta^4\mathbf{U}^n]|_{i,j,k} \\
+ \Delta\tau[\mathbf{H}^*]|_{i,j,k}.
\end{aligned}
\tag{4.27}
$$

where D_ξ , D_η and D_ζ are second-order accurate, central spatial differencing operators, defined in D_h. At each point in the mesh, $[\mathbf{RHS}]^n_{i,j,k}$ is a colunm vector with 5 components. Discretization of added higher order dissipation terms was already discussed in Sec.(4.4). Here we consider the computation of the first term in Eq.(4.27), using formulas given in Eq.(4.18) and Eq.(4.19):

$$
\begin{aligned}
[D_\xi(\mathbf{E}+\mathbf{T})^n + D_\eta(\mathbf{F}+\mathbf{V})^n + D_\zeta(\mathbf{G}+\mathbf{W})^n]|_{i,j,k} &= (1/2h_\xi)[\mathbf{E}(\mathbf{U}^n_{i+1,j,k}) - \mathbf{E}(\mathbf{U}^n_{i-1,j,k})] \\
&+ (1/h_\xi)\{\mathbf{T}[(\frac{\mathbf{U}^n_{i+1,j,k}+\mathbf{U}^n_{i,j,k}}{2}),(\frac{\mathbf{U}^n_{i+1,j,k}-\mathbf{U}^n_{i,j,k}}{h_\xi})] \\
&\quad - \mathbf{T}[(\frac{\mathbf{U}^n_{i,j,k}+\mathbf{U}^n_{i-1,j,k}}{2}),(\frac{\mathbf{U}^n_{i,j,k}-\mathbf{U}^n_{i-1,j,k}}{h_\xi})]\} \\
&+ (1/2h_\eta)[\mathbf{F}(\mathbf{U}^n_{i,j+1,k}) - \mathbf{F}(\mathbf{U}^n_{i,j-1,k})] \\
&+ (1/h_\eta)\{\mathbf{V}[(\frac{\mathbf{U}^n_{i,j+1,k}+\mathbf{U}^n_{i,j,k}}{2}),(\frac{\mathbf{U}^n_{i,j+1,k}-\mathbf{U}^n_{i,j,k}}{h_\eta})] \\
&\quad - \mathbf{V}[(\frac{\mathbf{U}^n_{i,j,k}+\mathbf{U}^n_{i,j-1,k}}{2}),(\frac{\mathbf{U}^n_{i,j,k}-\mathbf{U}^n_{i,j-1,k}}{h_\eta})]\} \\
&+ (1/2h_\zeta)[\mathbf{G}(\mathbf{U}^n_{i,j,k+1}) - \mathbf{G}(\mathbf{U}^n_{i,j,k-1})] \\
&+ (1/h_\zeta)\{\mathbf{W}[(\frac{\mathbf{U}^n_{i,j,k+1}+\mathbf{U}^n_{i,j,k}}{2}),(\frac{\mathbf{U}^n_{i,j,k+1}-\mathbf{U}^n_{i,j,k}}{h_\zeta})] \\
&\quad - \mathbf{W}[(\frac{\mathbf{U}^n_{i,j,k}+\mathbf{U}^n_{i,j,k-1}}{2}),(\frac{\mathbf{U}^n_{i,j,k}-\mathbf{U}^n_{i,j,k-1}}{h_\zeta})]\},
\end{aligned}
\tag{4.28}
$$

for $\{(i,j,k) \in D_h\}$. Also, $[\mathbf{RHS}]^n_{i,j,k} = 0$ for $i = 1$ or N_ξ, $j = 1$ or N_η and $k = 1$ or N_ζ.

This right hand side computation is equivalent in terms of both arithmetic and communication complexity to a regular sparse block (5×5) matrix-vector multiplication, which is the kernel (c). However, its efficient implementation, in this case, does not necessarily require the explicit formation and/or storage of the regular sparse matrix.

4.6. COMPUTATION OF THE FORCING VECTOR FUNCTION

Given the analytical form of $\mathbf{U}^*$, the forcing vector function $\mathbf{H}^*$ can simply be evaluated analytically as a function of ξ, η, and ζ, using Eq.(4.26). This function, along with Eq.(4.15), can then be used to evaluate $[\mathbf{H}^*]_{i,j,k}$, which is also a colunm vector with 5 components.

Here, we opt for a numerical evaluation of $[\mathbf{H}^*]_{i,j,k}$, using $[\mathbf{U}^*]_{i,j,k}$ and the finite-difference approximations of Eq.(4.18) and (4.19), as in the case of Eq.(4.28).

$$
\begin{aligned}
[\mathbf{H}^*]|_{i,j,k} \approx\ & (1/2h_\xi)[\mathbf{E}(\mathbf{U}^*_{i+1,j,k}) - \mathbf{E}(\mathbf{U}^*_{i-1,j,k})] \\
& + (1/h_\xi)\{\mathbf{T}[(\frac{\mathbf{U}^*_{i+1,j,k} + \mathbf{U}^*_{i,j,k}}{2}), (\frac{\mathbf{U}^*_{i+1,j,k} - \mathbf{U}^*_{i,j,k}}{h_\xi})] \\
& \quad - \mathbf{T}[(\frac{\mathbf{U}^*_{i,j,k} + \mathbf{U}^*_{i-1,j,k}}{2}), (\frac{\mathbf{U}^*_{i,j,k} - \mathbf{U}^*_{i-1,j,k}}{h_\xi})]\} \\
& + (1/2h_\eta)[\mathbf{F}(\mathbf{U}^*_{i,j+1,k}) - \mathbf{F}(\mathbf{U}^*_{i,j-1,k})] \\
& + (1/h_\eta)\{\mathbf{V}[(\frac{\mathbf{U}^*_{i,j+1,k} + \mathbf{U}^*_{i,j,k}}{2}), (\frac{\mathbf{U}^*_{i,j+1,k} - \mathbf{U}^*_{i,j,k}}{h_\eta})] \\
& \quad - \mathbf{V}[(\frac{\mathbf{U}^*_{i,j,k} + \mathbf{U}^*_{i,j-1,k}}{2}), (\frac{\mathbf{U}^*_{i,j,k} - \mathbf{U}^*_{i,j-1,k}}{h_\eta})]\} \\
& + (1/2h_\zeta)[\mathbf{G}(\mathbf{U}^*_{i,j,k+1}) - \mathbf{G}(\mathbf{U}^*_{i,j,k-1})] \\
& + (1/h_\zeta)\{\mathbf{W}[(\frac{\mathbf{U}^*_{i,j,k+1} + \mathbf{U}^*_{i,j,k}}{2}), (\frac{\mathbf{U}^*_{i,j,k+1} - \mathbf{U}^*_{i,j,k}}{h_\zeta})] \\
& \quad - \mathbf{W}[(\frac{\mathbf{U}^*_{i,j,k} + \mathbf{U}^*_{i,j,k-1}}{2}), (\frac{\mathbf{U}^*_{i,j,k} - \mathbf{U}^*_{i,j,k-1}}{h_\zeta})]\}, \\
& + \varepsilon\,[h^4_\xi D^4_\xi \mathbf{U}^* + h^4_\eta D^4_\eta \mathbf{U}^* + h^4_\zeta D^4_\zeta \mathbf{U}^*]|_{i,j,k},
\end{aligned}
\tag{4.29}
$$

for $\{(i,j,k) \in D_h\}$. The fourth difference dissipation terms are evaluated using Eq.(4.24). Also, $[\mathbf{H}]^*_{i,j,k} = 0$ for $i = 1$ or N_ξ, $j = 1$ or N_η and $k = 1$ or N_ζ.

4.7. SOLUTION OF THE SYSTEM OF LINEAR EQUATIONS

Replacing the spatial derivatives appearing in Eq.(4.25) by their equivalent difference approximations results in a system of linear equations for $[\Delta\mathbf{U}^n]_{i,j,k}$ for $i \in [2, N_\xi - 1]$, $j \in [2, N_\eta - 1]$ and $k \in [2, N_\zeta - 1]$. Direct solution of this system of linear equations requires a formidable matrix inversion effort, in terms of both the processing time and storage requirements. Therefore, $\Delta\mathbf{U}^n$ is obtained through the use of an iterative method. Here we consider three such iterative methods, involving the kernels (a), (b) and (d). All methods involve some form of approximation to the implicit operator or the **LHS** of Eq.(4.25). For pseudo-time marching schemes, it is generally sufficient to perform only one iteration per time step.

4.7.1. APPROXIMATE FACTORIZATION (BEAM-WARMING) ALGORITHM

In this method, the implicit operator in Eq.(4.25a) is approximately factored in the following manner, ([1],[9]):

$$
\{\mathbf{I} - \Delta\tau[\frac{\partial(\mathbf{A})^n}{\partial\xi} + \frac{\partial^2(\mathbf{N})^n}{\partial\xi^2}]\} \times \{I - \Delta\tau[\frac{\partial(\mathbf{B})^n}{\partial\eta} + \frac{\partial^2(\mathbf{Q})^n}{\partial\eta^2}]\} \times \{I - \Delta\tau[\frac{\partial(\mathbf{C})^n}{\partial\zeta} + \frac{\partial^2(\mathbf{S})^n}{\partial\zeta^2}]\}\Delta\mathbf{U}^n = \mathbf{RHS}
\tag{4.30}
$$

The Beam-Warming algorithm is to be implemented as shown below:

INITIALIZATION:

Set the boundary values of $\mathbf{U}_{i,j,k}$ for $(i,j,k) \in \partial D_h$ in accordance with Eq.(3.7).

Set the initial values of $\mathbf{U}^0_{i,j,k}$ for $(i,j,k) \in D_h$ in accordance with Eq.(3.8).

Compute the forcing function vector, $\mathbf{H}^*_{i,j,k}$ for $(i,j,k) \in D_h$, using Eq.(4.29).

STEP 1: EXPLICIT PART.

Compute the $[\mathbf{RHS}]^n_{i,j,k}$ for $(i,j,k) \in D_h$.

STEP 2: ξ-SWEEP OF THE IMPLICIT PART.

Form and solve the following system of linear equations for $[\Delta\mathbf{U}_1]_{i,j,k}$ for $(i,j,k) \in D_h$:

$$\{\mathbf{I} - \Delta\tau[D_\xi(\mathbf{A})^n + D^2_\xi(\mathbf{N})^n]\}\Delta\mathbf{U}_1 = \mathbf{RHS}.$$

STEP 3: η-SWEEP OF THE IMPLICIT PART.

Form and solve the following system of linear equations for $[\Delta\mathbf{U}_2]_{i,j,k}$ for $(i,j,k) \in D_h$:

$$\{\mathbf{I} - \Delta\tau[D_\eta(\mathbf{B})^n + D^2_\eta(\mathbf{Q})^n]\}\Delta\mathbf{U}_2 = \Delta\mathbf{U}_1$$

STEP 4: ζ-SWEEP OF THE IMPLICIT PART.

Form and solve the following system of linear equations for $[\Delta\mathbf{U}^n]_{i,j,k}$ for $(i,j,k) \in D_h$:

$$\{\mathbf{I} - \Delta\tau[D_\zeta(\mathbf{C})^n + D^2_\zeta(\mathbf{S})^n]\}\Delta\mathbf{U}^n = \Delta\mathbf{U}_2.$$

STEP 5: UPDATE THE SOLUTION.

$$[\mathbf{U}^{n+1}]_{i,j,k} = [\mathbf{U}^n]_{i,j,k} + [\Delta\mathbf{U}^n]_{i,j,k}, \qquad \text{for} \quad (\mathrm{i,j,k}) \in \mathrm{D_h}$$

Steps (1) - (5) consists of one time-stepping iteration of the Approximate Factorization scheme. The solution of systems of linear equations in each of the steps (2) - (4) is equivalent to the solution of multiple, independent systems of block tridiagonal equations, with each block being a (5×5) matrix, in the three coordinate directions ξ, η, ζ respectively. For example, the system of equations in the ξ-sweep has the following block tridiagonal structure:

$$\begin{aligned}
[\mathcal{B}_{1,j,k}][\Delta\mathbf{U}_1]_{1,j,k} + [\mathcal{C}_{1,j,k}][\Delta\mathbf{U}_1]_{2,j,k} &= [\mathbf{RHS}]_{1,j,k}, \\
[\mathcal{A}_{i,j,k}][\Delta\mathbf{U}_1]_{i-1,j,k} + [\mathcal{B}_{i,j,k}][\Delta\mathbf{U}_1]_{i,j,k} + [\mathcal{C}_{i,j,k}][\Delta\mathbf{U}_1]_{i+1,j,k} &= [\mathbf{RHS}]_{i,j,k}; \qquad 2 \le i \le N_\xi - 1. \\
[\mathcal{A}_{N_\xi,j,k}][\Delta\mathbf{U}_1]_{N_\xi-1,j,k} + [\mathcal{B}_{N_\xi,j,k}][\Delta\mathbf{U}_1]_{N_\xi,j,k} &= [\mathbf{RHS}]_{N_\xi,j,k}.
\end{aligned} \tag{4.31}$$

where, $(j \in [2, N_\eta - 1])$ and $(k \in [2, N_\zeta - 1])$. Also, $[\mathcal{A}]$, $[\mathcal{B}]$ and $[\mathcal{C}]$ are (5×5) matrices and $[\Delta \mathbf{U}_1]_{i,j,k}$ is a (5×1) column vector.

Here, for $2 \leq i \leq (N_\xi - 1)$, using Eq.(4.20) and (4.22):

$$\begin{aligned}
[\mathcal{A}_{i,j,k}] &= -\Delta\tau\{(-1/2h_\xi)[\mathbf{A}(\mathbf{U}^n_{i-1,j,k})] + (1/h_\xi^2)[\mathbf{N}(\mathbf{U}^n_{i-1,j,k})]\}, \\
[\mathcal{B}_{i,j,k}] &= \mathbf{I} + \Delta\tau(2/h_\xi^2)[\mathbf{N}(\mathbf{U}^n_{i,j,k})], \\
[\mathcal{C}_{i,j,k}] &= -\Delta\tau\{(1/2h_\xi)[\mathbf{A}(\mathbf{U}^n_{i+1,j,k})] + (1/h_\xi^2)[\mathbf{N}(\mathbf{U}^n_{i+1,j,k})]\}.
\end{aligned} \tag{4.32}$$

Also, $[\mathcal{B}_{1,j,k}] = [\mathbf{I}], [\mathcal{C}_{1,j,k}] = [0], [\mathcal{A}_{N_\xi,j,k}] = [0]$, and $[\mathcal{B}_{N_\xi,j,k}] = [\mathbf{I}]$.

4.7.2. DIAGONAL FORM OF THE APPROXIMATE FACTORIZATION ALGORITHM

Here the approximate factorization algorithm of Sec.(4.7.1) is modified so as to transform the coupled systems given by the left hand side of Eq.(4.25b), into an uncoupled diagonal form. This involves further approximations in the treatment of the implicit operator. This diagonalization process targets only the matrices **A**, **B**, and **C** in the implicit operator. Effects of other matrices present in the implicit operator are either ignored or approximated to conform to the resulting diagonal structure.

The diagonalization process is based on the observation that matrices **A**, **B** and **C**, each have a set of real eigenvalues and a complete set of eigenvectors. Therefore, they can be diagonalized through similarity transformations ([13]):

$$\begin{aligned}
\mathbf{A} &= \mathbf{T}_\xi \Lambda_\xi \mathbf{T}_\xi^{-1}; \\
\mathbf{B} &= \mathbf{T}_\eta \Lambda_\eta \mathbf{T}_\eta^{-1}; \\
\mathbf{C} &= \mathbf{T}_\zeta \Lambda_\zeta \mathbf{T}_\zeta^{-1}
\end{aligned} \tag{4.33}$$

with,

$$\begin{aligned}
\Lambda_\xi &= Diag\,[-(u^{(2)}/u^{(1)}),\ -(u^{(2)}/u^{(1)}),\ -(u^{(2)}/u^{(1)}),\ -(u^{(2)}/u^{(1)} + a),\ -(u^{(2)}/u^{(1)} - a)]; \\
\Lambda_\eta &= Diag\,[-(u^{(3)}/u^{(1)}),\ -(u^{(3)}/u^{(1)}),\ -(u^{(3)}/u^{(1)}),\ -(u^{(3)}/u^{(1)} + a),\ -(u^{(3)}/u^{(1)} - a)]; \\
\Lambda_\zeta &= Diag\,[-(u^{(4)}/u^{(1)}),\ -(u^{(4)}/u^{(1)}),\ -(u^{(4)}/u^{(1)}),\ -(u^{(4)}/u^{(1)} + a),\ -(u^{(4)}/u^{(1)} - a)],
\end{aligned} \tag{4.34}$$

where,

$$a = \sqrt{c_1\, c_2\{\frac{u^{(5)}}{u^{(1)}} - 0.5\,[\frac{[u^{(2)}]^2 + [u^{(3)}]^2 + [u^{(4)}]^2}{[u^{(1)}]^2}]\}} \tag{4.35}$$

and $\mathbf{T}_\xi(\mathbf{U})$, $\mathbf{T}_\eta(\mathbf{U})$ and $\mathbf{T}_\zeta(\mathbf{U})$ are the matrices whose columns are the eigenvectors of **A**, **B** and **C** respectively. When all other matrices except **A**, **B** and **C** are ignored, the implicit operator is given by:

$$\mathbf{LHS} = [I - \Delta\tau\frac{\partial \mathbf{A}^n}{\partial \xi}][I - \Delta\tau\frac{\partial \mathbf{B}^n}{\partial \eta}][I - \Delta\tau\frac{\partial \mathbf{C}^n}{\partial \zeta}] \tag{4.36}$$

Substituting for $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$, using Eq.(4.33) in Eq. (4.36), we get:

$$\begin{aligned}\mathbf{LHS} =&[(\mathbf{T}_\xi\mathbf{T}_\xi^{-1})^n - \Delta\tau\frac{\partial(\mathbf{T}_\xi\Lambda_\xi\mathbf{T}_\xi^{-1})^n}{\partial\xi}] \\ &\times[(\mathbf{T}_\eta\mathbf{T}_\eta^{-1})^n - \Delta\tau\frac{\partial(\mathbf{T}_\eta\Lambda_\eta\mathbf{T}_\eta^{-1})^n}{\partial\eta}]\times[(\mathbf{T}_\zeta\mathbf{T}_\zeta^{-1})^n - \Delta\tau\frac{\partial(\mathbf{T}_\zeta\Lambda_\zeta\mathbf{T}_\zeta^{-1})^n}{\partial\zeta}]\Delta\mathbf{U}^n\end{aligned} \tag{4.37}$$

A modified form of Eq.(4.37) is obtained by moving the eigenvector matrices $\mathbf{T}_\xi$, $\mathbf{T}_\eta$ and $\mathbf{T}_\zeta$ outside the spatial differential operators $\partial/\partial\xi$, $\partial/\partial\eta$ and $\partial/\partial\zeta$ respectively, ([13]):

$$\mathbf{LHS} = \mathbf{T}_\xi^n[\mathbf{I} - \Delta\tau\frac{\partial(\Lambda_\xi)^n}{\partial\xi}](\mathbf{T}_\xi^{-1})^n(\mathbf{T}_\eta)^n[\mathbf{I} - \Delta\tau\frac{\partial(\Lambda_\eta)^n}{\partial\eta}](\mathbf{T}_\eta^{-1})^n(\mathbf{T}_\zeta)^n[\mathbf{I} - \Delta\tau\frac{\partial(\Lambda_\zeta)^n}{\partial\zeta}](\mathbf{T}_\zeta^{-1})^n\Delta\mathbf{U}^n$$

This can be written as:

$$\mathbf{LHS} = \mathbf{T}_\xi^n[\mathbf{I} - \Delta\tau\frac{\partial(\Lambda_\xi)^n}{\partial\xi}]\bar{\mathbf{N}}[\mathbf{I} - \Delta\tau\frac{\partial(\Lambda_\eta)^n}{\partial\eta}]\bar{\mathbf{P}}[\mathbf{I} - \Delta\tau\frac{\partial(\Lambda_\zeta)^n}{\partial\zeta}](\mathbf{T}_\zeta^{-1})^n\Delta\mathbf{U}^n$$

where,

$$\begin{aligned}\bar{\mathbf{N}} &= \mathbf{T}_\xi^{-1}\mathbf{T}_\eta; \qquad \bar{\mathbf{N}}^{-1} = \mathbf{T}_\eta^{-1}\mathbf{T}_\xi. \\ \bar{\mathbf{P}} &= \mathbf{T}_\eta^{-1}\mathbf{T}_\zeta; \qquad \bar{\mathbf{P}}^{-1} = \mathbf{T}_\zeta^{-1}\mathbf{T}_\eta.\end{aligned} \tag{3.38}$$

The eigenvector matrices are functions of ξ, η and ζ and therefore this modification introduces further errors of $O(\Delta\tau)$ into the factorization process.

During the factorization process, presence of the matrices **N**,**Q** and **S** in the implicit part were ignored. This is because, in general, the similarity transformations used for diagonalizing **A** do not simultaneously diagonalize **N**. The same is true for the η and ζ factors as well. This necessitates some ad-hoc approximate treatment of the ignored terms, which at the same time preserves the diagonal structure of the implicit operators. Whatever is the approach used, additional approximation errors are introduced in the treatment of the implicit operators. We chose to approximate $\mathbf{N}, \mathbf{Q}$ and **S** by diagonal matrices in the implicit operators, whose values are given by the the spectral radii $\rho(\mathbf{N})$, $\rho(\mathbf{Q})$ and $\rho(\mathbf{S})$ respectively ([17]). In addition, we also treat the added fourth-difference dissipation terms implicitly.

$$\begin{aligned}\mathbf{LHS} = \mathbf{T}_\xi^n[\mathbf{I} - \Delta\tau\{&\frac{\partial(\Lambda_\xi)^n}{\partial\xi} + \frac{\partial^2[\rho(\mathbf{N})^n\mathbf{I}]}{\partial\xi^2} - \varepsilon\, h_\xi^4\frac{\partial^4(\mathbf{I})}{\partial\xi^4}\}] \\ &\times\bar{\mathbf{N}}[\mathbf{I} - \Delta\tau\{\frac{\partial(\Lambda_\eta)^n}{\partial\eta} + \frac{\partial^2[\rho(\mathbf{Q})^n\mathbf{I}]}{\partial\eta^2} - \varepsilon\, h_\eta^4\frac{\partial^4(\mathbf{I})}{\partial\eta^4}\}] \\ &\times\bar{\mathbf{P}}[\mathbf{I} - \Delta\tau\{\frac{\partial(\Lambda_\zeta)^n}{\partial\zeta} + \frac{\partial^2[\rho(\mathbf{S})^n\mathbf{I}]}{\partial\zeta^2} - \varepsilon\, h_\zeta^4\frac{\partial^4(\mathbf{I})}{\partial\zeta^4}\}]\mathbf{T}_\zeta^{-1}\Delta\mathbf{U}^n.\end{aligned} \tag{4.39}$$

The matrices $\mathbf{T}_\xi^{-1}$,$\mathbf{T}_\zeta$,$\bar{\mathbf{N}}^{-1}$ and $\bar{\mathbf{P}}^{-1}$ are given by:

$$\mathbf{T}_\xi^{-1} = \begin{pmatrix} (1-[q/a^2]) & c_2[u^{(2)}/u^{(1)}]/a^2 & c_2[u^{(3)}/u^{(1)}]/a^2 & c_2[u^{(4)}/u^{(1)}]/a^2 & -[c_2/a^2] \\ -(u^{(4)}/[u^{(1)}]^2) & 0 & 0 & (1/u^{(1)}) & 0 \\ (u^{(3)}/[u^{(1)}]^2) & 0 & -(1/u^{(1)}) & 0 & 0 \\ \sigma(q - a[u^{(2)}/u^{(1)}]) & \sigma(a - c_2[u^{(2)}/u^{(1)}]) & -\sigma c_2[u^{(3)}/u^{(1)}] & -\sigma c_2[u^{(4)}/u^{(1)}] & \sigma c_2 \\ \sigma(q + a[u^{(2)}/u^{(1)}]) & -\sigma(a + c_2[u^{(2)}/u^{(1)}]) & -\sigma c_2[u^{(3)}/u^{(1)}] & -\sigma c_2[u^{(4)}/u^{(1)}] & \sigma c_2 \end{pmatrix}$$

$$\mathbf{T}_\zeta = \begin{pmatrix} 0 & 0 & 1 & \alpha & \alpha \\ 0 & -u^{(1)} & [u(2)/u(1)] & \alpha[u(2)/u(1)] & \alpha[u(2)/u(1)] \\ u^{(1)} & 0 & [u(3)/u(1)] & \alpha[u(3)/u(1)] & \alpha[u(3)/u(1)] \\ 0 & 0 & [u(4)/u(1)] & \alpha([u(4)/u(1)]+a) & \alpha([u(4)/u(1)]-a) \\ u^{(3)} & -u^{(2)} & [q/c_2] & \alpha[(q+a^2)/c_2 + a(u^{(4)}/u^{(1)})] & \alpha[(q+a^2)/c_2 - a(u^{(4)}/u^{(1)})] \end{pmatrix},$$

$$\tilde{\mathbf{N}}^{-1} = \begin{pmatrix} 0 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/\sqrt{2} & -1/\sqrt{2} \\ 0 & 0 & -1/\sqrt{2} & 1/2 & 1/2 \\ 0 & 0 & 1/\sqrt{2} & 1/2 & 1/2 \end{pmatrix},$$

$$\tilde{\mathbf{P}}^{-1} = \begin{pmatrix} 0 & 0 & 0 & 1/\sqrt{2} & -1/\sqrt{2} \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ -1/\sqrt{2} & 0 & 0 & 1/2 & 1/2 \\ 1/\sqrt{2} & 0 & 0 & 1/2 & 1/2 \end{pmatrix},$$

where $\sigma = 1/(\sqrt{2}\,u^{(1)}a)$ and $\alpha = [u^{(1)}/(\sqrt{2}\,a)]$.

In addition, the spectral radii of the matrices $\mathbf{N}, \mathbf{Q}$ and $\mathbf{S}$ are given by:

$$\begin{aligned} \rho(\mathbf{N}) = max(& d_\xi^{(1)}, \\ & d_\xi^{(2)} + [4./3.]k_3k_4[1./u^{(1)}], \\ & d_\xi^{(3)} + k_3k_4[1./u^{(1)}], \\ & d_\xi^{(4)} + k_3k_4[1./u^{(1)}], \\ & d_\xi^{(5)} + k_1k_3k_4k_5[1./u^{(1)}]\ ), \end{aligned}$$

$$\begin{aligned} \rho(\mathbf{Q}) = max(& d_\eta^{(1)}, \\ & d_\eta^{(2)} + k_3k_4[1./u^{(1)}], \\ & d_\eta^{(3)} + [4./3.]k_3k_4[1./u^{(1)}], \\ & d_\eta^{(4)} + k_3k_4[1./u^{(1)}], \\ & d_\eta^{(5)} + k_1k_3k_4k_5[1./u^{(1)}]\ ), \end{aligned}$$

$$\begin{aligned} \rho(\mathbf{S}) = max(& d_\zeta^{(1)}, \\ & d_\zeta^{(2)} + k_3k_4[1./u^{(1)}], \\ & d_\zeta^{(3)} + k_3k_4[1./u^{(1)}], \\ & d_\zeta^{(4)} + [4./3.]k_3k_4[1./u^{(1)}], \\ & d_\zeta^{(5)} + k_1k_3k_4k_5[1./u^{(1)}]\ ). \end{aligned}$$

The explicit part of the diagonal algorithm is exactly the same as in Eq.(4.26b) and all the approximations are restricted to the implicit operator. Therefore, if the diagonal algorithm converges, the steady-state solution will be identical to the one obtained without diagonalization. However, the convergence behavior of the diagonalized algorithm would be different.

The Diagonalized Approximate Factorization algorithm is to be implemented in the following order:

INITIALIZATION:

Set the boundary values of $\mathbf{U}_{i,j,k}$ for $(i,j,k) \in \partial D_h$ in accordance with Eq.(3.7).
Set the initial values of $\mathbf{U}^0_{i,j,k}$ for $(i,j,k) \in D_h$ in accordance with Eq.(3.8).
Compute the forcing function vector, $\mathbf{H}^*_{i,j,k}$ for $(i,j,k) \in D_h$, using Eq.(4.29).

STEP 1: EXPLICIT PART.

Compute $[\mathbf{RHS}]^n_{i,j,k}$ for $(i,j,k) \in D_h$.

STEP 2:

Perform the matrix-vector multiplication:

$$[\Delta \mathbf{U}_1] = (\mathbf{T}_\xi^{-1})^n [\mathbf{RHS}].$$

STEP 3: ξ-SWEEP OF THE IMPLICIT PART.

Form and solve the following system of linear equations for $\Delta \mathbf{U}_2$:

$$\{I - \Delta\tau[D_\xi(\Lambda_\xi)^n] - \Delta\tau[D_\xi^2(\rho(\mathbf{N})^n\mathbf{I})] + \Delta\tau[\varepsilon\, h_\xi^4 D_\xi^4(\mathbf{I})]\}[\Delta \mathbf{U}_2] = [\Delta \mathbf{U}_1].$$

STEP 4:

Perform the matrix-vector multiplication:

$$[\Delta \mathbf{U}_3] = \bar{\mathbf{N}}^{-1}[\Delta \mathbf{U}_2].$$

STEP 5: η-SWEEP OF THE IMPLICIT PART.

Form and solve the following system of linear equations for $\Delta \mathbf{U}_4$:

$$\{I - \Delta\tau[D_\eta(\Lambda_\eta)^n] - \Delta\tau[D_\eta^2(\rho(\mathbf{Q})^n\mathbf{I})] + \Delta\tau[\varepsilon\, h_\eta^4 D_\eta^4(\mathbf{I})]\}[\Delta \mathbf{U}_4] = [\Delta \mathbf{U}_3].$$

STEP 6:

Perform the matrix-vector multiplication:

$$[\Delta \mathbf{U}_5] = \bar{\mathbf{P}}^{-1}[\Delta \mathbf{U}_4].$$

STEP 7: ζ-SWEEP OF THE IMPLICIT PART.

Form and solve the following system of linear equations for $\Delta \mathbf{U}_6$:

$$\{I - \Delta\tau[D_\zeta(\Lambda_\zeta)^n] - \Delta\tau[D_\zeta^2(\rho(\mathbf{S})^n\mathbf{I})] + \Delta\tau[\varepsilon\, h_\zeta^4 D_\zeta^4(\mathbf{I})]\}[\Delta \mathbf{U}_6] = [\Delta \mathbf{U}_5].$$

STEP 8:

Perform the matrix-vector multiplication:

$$[\Delta \mathbf{U}^n] = \mathbf{T}_\zeta[\Delta \mathbf{U}_6].$$

STEP 9: UPDATE THE SOLUTION.

$$\mathbf{U}^{n+1} = \mathbf{U}^n + \Delta \mathbf{U}^n.$$

Steps (1) - (9) constitute of one iteration of the Diagonal Form of the Approximate Factorization algorithm. The new implicit operators are block pentadiagonal. However, the blocks are diagonal in form, so that the operators simplify into five independent scalar pentadiagonal systems. Therefore, each of the steps (3), (5) and (7) involve the solution of multiple, independent systems of scalar pentadiagonal equations in the three coordinate directions ξ, η and ζ respectively. For example, each of the five independent scalar pentadiagonal systems in the ξ-sweep has the following structure:

$$\begin{aligned}
\mathrm{c}_{1,j,k}[\Delta \mathbf{U}_2]^{(m)}_{1,j,k} + \mathrm{d}_{1,j,k}[\Delta \mathbf{U}_2]^{(m)}_{2,j,k} + \mathrm{e}_{1,j,k}[\Delta \mathbf{U}_2]^{(m)}_{3,j,k} &= [\Delta \mathbf{U}_1]^{(m)}_{1,j,k}, \\
\mathrm{b}_{2,j,k}[\Delta \mathbf{U}_2]^{(m)}_{1,j,k} + \mathrm{c}_{2,j,k}[\Delta \mathbf{U}_2]^{(m)}_{2,j,k} + \mathrm{d}_{2,j,k}[\Delta \mathbf{U}_2]^{(m)}_{3,j,k} + \mathrm{e}_{2,j,k}[\Delta \mathbf{U}_2]^{(m)}_{4,j,k} &= [\Delta \mathbf{U}_1]^{(m)}_{2,j,k}, \\
\mathrm{a}_{i,j,k}[\Delta \mathbf{U}_2]^{(m)}_{i-2,j,k} + \mathrm{b}_{i,j,k}[\Delta \mathbf{U}_2]^{(m)}_{i-1,j,k} + \mathrm{c}_{i,j,k}[\Delta \mathbf{U}_2]^{(m)}_{i,j,k} + \mathrm{d}_{i,j,k}[\Delta \mathbf{U}_2]^{(m)}_{i+1,j,k} & \\
+\mathrm{e}_{i,j,k}[\Delta \mathbf{U}_2]^{(m)}_{i+2,j,k} = [\Delta \mathbf{U}_1]^{(m)}_{i,j,k}, \quad \text{for } 3 \le \mathrm{i} \le (N_\xi - 2) & \\
\mathrm{a}_{N_\xi-1,j,k}[\Delta \mathbf{U}_2]^{(m)}_{N_\xi-3,j,k} + \mathrm{b}_{N_\xi-1,j,k}[\Delta \mathbf{U}_2]^{(m)}_{N_\xi-2,j,k} + \mathrm{c}_{N_\xi-1,j,k}[\Delta \mathbf{U}_2]^{(m)}_{N_\xi-1,j,k} & \\
+\mathrm{d}_{N_\xi-1,j,k}[\Delta \mathbf{U}_2]^{(m)}_{N_\xi,j,k} = [\Delta \mathbf{U}_1]^{(m)}_{N_\xi-1,j,k}, & \\
\mathrm{a}_{N_\xi,j,k}[\Delta \mathbf{U}_2]^{(m)}_{N_\xi-2,j,k} + \mathrm{b}_{N_\xi,j,k}[\Delta \mathbf{U}_2]^{(m)}_{N_\xi-1,j,k} + \mathrm{c}_{N_\xi,j,k}[\Delta \mathbf{U}_2]^{(m)}_{N_\xi,j,k} &= [\Delta \mathbf{U}_1]^{(m)}_{N_\xi,j,k}.
\end{aligned} \tag{4.40}$$

where, $(j \in [2, N_\eta - 1])$, $(k \in [2, N_\zeta - 1])$ and $(m \in [1, 5])$. The elements of the pentadiagonal matrix are given by the following:

$$\mathrm{c}_{1,j,k} = 1.; \qquad \mathrm{d}_{1,j,k} = 0.; \qquad \mathrm{e}_{1,j,k} = 0. \tag{3.41a}$$

$$\begin{aligned}
\mathrm{b}_{2,j,k} &= \Delta\tau(1/2h_\xi)[(\Lambda_\xi)^n]^{(m,m)}_{1,j,k} - \Delta\tau(1/h_\xi^2)[\rho(\mathrm{N})^n]_{1,j,k}, \\
\mathrm{c}_{2,j,k} &= 1. + \Delta\tau(2/h_\xi^2)[\rho(\mathrm{N})^n]_{2,j,k} + \Delta\tau\,\varepsilon\,(5), \\
\mathrm{d}_{2,j,k} &= -\Delta\tau(1/2h_\xi)[(\Lambda_\xi)^n]^{(m,m)}_{3,j,k} - \Delta\tau(1/h_\xi^2)[\rho(\mathrm{N})^n]_{3,j,k} + \Delta\tau\,\varepsilon\,(-4), \\
\mathrm{e}_{2,j,k} &= \Delta\tau\,\varepsilon.
\end{aligned} \tag{4.41b}$$

$$
\begin{aligned}
\mathbf{a}_{3,j,k} &= 0.0, \\
\mathbf{b}_{3,j,k} &= \Delta\tau(1/2h_\xi)[(\Lambda_\xi)^n]^{(m,m)}_{2,j,k} - \Delta\tau(1/h_\xi^2)[\rho(\mathbf{N})^n]_{2,j,k} + \Delta\tau\,\varepsilon\,(-4), \\
\mathbf{c}_{3,j,k} &= 1. + \Delta\tau(2/h_\xi^2)[\rho(\mathbf{N})^n]_{3,j,k} + \Delta\tau\,\varepsilon\,(6), \\
\mathbf{d}_{3,j,k} &= -\Delta\tau(1/2h_\xi)[(\Lambda_\xi)^n]^{(m,m)}_{4,j,k} - \Delta\tau(1/h_\xi^2)[\rho(\mathbf{N})^n]_{4,j,k} + \Delta\tau\,\varepsilon\,(-4), \\
\mathbf{e}_{3,j,k} &= \Delta\tau\,\varepsilon.
\end{aligned}
\tag{4.41c}
$$

$$
\begin{aligned}
\mathbf{a}_{i,j,k} &= \Delta\tau\,\varepsilon, \\
\mathbf{b}_{i,j,k} &= \Delta\tau(1/2h_\xi)[(\Lambda_\xi)^n]^{(m,m)}_{i-1,j,k} - \Delta\tau(1/h_\xi^2)[\rho(\mathbf{N})^n]_{i-1,j,k} + \Delta\tau\,\varepsilon\,(-4), \\
\mathbf{c}_{i,j,k} &= 1. + \Delta\tau(2/h_\xi^2)[\rho(\mathbf{N})^n]_{i,j,k} + \Delta\tau\,\varepsilon\,(6), \\
\mathbf{d}_{i,j,k} &= -\Delta\tau(1/2h_\xi)[(\Lambda_\xi)^n]^{(m,m)}_{i+1,j,k} - \Delta\tau(1/h_\xi^2)[\rho(\mathbf{N})^n]_{i+1,j,k} + \Delta\tau\,\varepsilon\,(-4), \\
\mathbf{e}_{i,j,k} &= \Delta\tau\,\varepsilon.
\end{aligned}
\tag{4.41d}
$$

$$
\begin{aligned}
\mathbf{a}_{N_\ell-2,j,k} &= \Delta\tau\,\varepsilon, \\
\mathbf{b}_{N_\ell-2,j,k} &= \Delta\tau(1/2h_\xi)[(\Lambda_\xi)^n]^{(m,m)}_{N_\ell-3,j,k} - \Delta\tau(1/h_\xi^2)[\rho(\mathbf{N})^n]_{N_\ell-3,j,k} + \Delta\tau\,\varepsilon\,(-4), \\
\mathbf{c}_{N_\ell-2,j,k} &= 1. + \Delta\tau(2/h_\xi^2)[\rho(\mathbf{N})^n]_{N_\ell-2,j,k} + \Delta\tau\,\varepsilon\,(6), \\
\mathbf{d}_{N_\ell-2,j,k} &= -\Delta\tau(1/2h_\xi)[(\Lambda_\xi)^n]^{(m,m)}_{N_\ell-1,j,k} - \Delta\tau(1/h_\xi^2)[\rho(\mathbf{N})^n]_{N_\ell-1,j,k} + \Delta\tau\,\varepsilon\,(-4), \\
\mathbf{e}_{N_\ell-2,j,k} &= 0.0.
\end{aligned}
\tag{4.41e}
$$

$$
\begin{aligned}
\mathbf{a}_{N_\ell-1,j,k} &= \Delta\tau\,\varepsilon, \\
\mathbf{b}_{N_\ell-1,j,k} &= \Delta\tau(1/2h_\xi)[(\Lambda_\xi)^n]^{(m,m)}_{N_\ell-2,j,k} - \Delta\tau(1/h_\xi^2)[\rho(\mathbf{N})^n]_{N_\ell-2,j,k} + \Delta\tau\,\varepsilon\,(-4), \\
\mathbf{c}_{N_\ell-1,j,k} &= 1. + \Delta\tau(2/h_\xi^2)[\rho(\mathbf{N})^n]_{N_\ell-1,j,k} + \Delta\tau\,\varepsilon\,(5), \\
\mathbf{d}_{N_\ell-1,j,k} &= -\Delta\tau(1/2h_\xi)[(\Lambda_\xi)^n]^{(m,m)}_{N_\ell,j,k} - \Delta\tau(1/h_\xi^2)[\rho(\mathbf{N})^n]_{N_\ell,j,k}.
\end{aligned}
\tag{4.41f}
$$

$$
\mathbf{a}_{N_\ell,j,k} = 0.; \qquad \mathbf{b}_{N_\ell,j,k} = 0.; \qquad \mathbf{c}_{N_\ell,j,k} = 1. \tag{4.41g}
$$

4.3.5. SYMMETRIC SUCCESSIVE OVER-RELAXATION ALGORITHM

In this method, the system of linear equations obtained after replacing the spatial derivatives appearing in the implicit operator in Eq.(4.25a) by second-order accurate, central finite difference operators, is solved using the symmetric, successive over-relaxation scheme. Let the linear system be given by:

$$
[\mathbf{K}^n][\Delta\mathbf{U}^n] = [\mathbf{RHS}^n] \tag{4.42}
$$

where,

$$
\mathbf{K}^n = \{\mathbf{I} - \Delta\tau[D_\xi\mathbf{A}^n + D_\xi^2\mathbf{N}^n + D_\eta\mathbf{B}^n + D_\eta^2\mathbf{Q}^n + D_\zeta\mathbf{C}^n + D_\zeta^2\mathbf{S}^n]\}\Delta\mathbf{U}^n, \quad \text{for } (i,j,k) \in D_h.
$$

It is specified that the unknowns be ordered corresponding to the gridpoints, lexicographically, such that the index in ξ-direction runs fastest, followed by the index in η-direction and finally in

ζ-direction. The finite-difference discretization matrix $\mathbf{K}$, resulting from such an ordering has a very regular, banded structure. There are altogether seven block diagonals, each with a (5×5) block size.

The matrix $\mathbf{K}$ can be written as the sum of the matrices $\mathbf{D}$, $\mathbf{Y}$ and $\mathbf{Z}$:

$$\mathbf{K}^n = \mathbf{D}^n + \mathbf{Y}^n + \mathbf{Z}^n, \tag{4.43}$$

where,

$$\mathbf{D}^n = \text{Main block} - \text{diagonal of } \mathbf{K}^n;$$

$$\mathbf{Y}^n = \text{Three sub} - \text{block} - \text{diagonals of } \mathbf{K}^n;$$

$$\mathbf{Z}^n = \text{Three super} - \text{block} - \text{diagonals of } \mathbf{K}^n.$$

Therefore, $\mathbf{D}$ is a block-diagonal matrix, while $\mathbf{Y}$ and $\mathbf{Z}$ are strictly lower and upper triangular, respectively. Then the point-SSOR iteration scheme can be written as ([14],[15]):

$$[\mathbf{X}^n][\Delta\mathbf{U}^n] = [\mathbf{RHS}] \tag{4.44}$$

where,

$$\begin{aligned} \mathbf{X}^n &= \omega(2.-\omega)(\mathbf{D}^n + \omega\mathbf{Y}^n)(\mathbf{D}^n)^{-1}(\mathbf{D}^n + \omega\mathbf{Z}^n), \\ &= \omega(2.-\omega)(\mathbf{D}^n + \omega\mathbf{Y}^n)(\mathbf{I} + \omega(\mathbf{D}^n)^{-1}\mathbf{Z}^n) \end{aligned} \tag{4.45}$$

and $\omega \in (0.,2.)$ is the over-relaxation factor (a specified constant). The SSOR algorithm is to be implemented in following manner:

INITIALIZATION:

Set the boundary values of $\mathbf{U}_{i,j,k}$ in accordance with Eq.(3.7).

Set the initial values of $\mathbf{U}^0_{i,j,k}$ in accordance with Eq.(3.8).

Compute the forcing function vector, $\mathbf{H}^*_{i,j,k}$. for $(i,j,k) \in D_h$, using Eq.(4.29).

STEP 1: THE EXPLICIT PART.

Compute $[\mathbf{RHS}]^n_{i,j,k}$ for $(i,j,k) \in D_h$.

STEP 2: LOWER TRIANGULAR SYSTEM SOLUTION.

Form and solve the following regular, sparse, block lower triangular system to get $[\Delta\mathbf{U}_1]$:

$$(\mathbf{D}^n + \omega\mathbf{Y}^n)[\Delta\mathbf{U}_1] = [\mathbf{RHS}].$$

STEP 3: UPPER TRIANGULAR SYSTEM SOLUTION.

Form and solve the following regular, sparse, block upper triangular system to get $[\Delta\mathbf{U}^n]$:

$$(\mathbf{I} + \omega(\mathbf{D}^n)^{-1}\mathbf{Z}^n)[\Delta\mathbf{U}^n] = [\Delta\mathbf{U}_1].$$

STEP 4:

Update the solution:

$$\mathbf{U}^{n+1} = \mathbf{U}^n + [1/\omega(2.-\omega)]\Delta\mathbf{U}^n.$$

Steps (1)-(4) constitute one complete iteration of the SSOR scheme. The l-th block row of the matrix $\mathbf{K}$ has the following structure:

$$\begin{aligned}[\mathcal{A}_l][\Delta\mathbf{U}^n]_{l-(N_\xi-2)(N_\eta-2)} + [\mathcal{B}_l][\Delta\mathbf{U}^n]_{l-(N_\xi-2)} + [\mathcal{C}_l][\Delta\mathbf{U}^n]_{l-1} + [\mathcal{D}_l][\Delta\mathbf{U}^n]_l \\ + [\mathcal{E}_l][\Delta\mathbf{U}^n]_{l+1} + [\mathcal{F}_l][\Delta\mathbf{U}^n]_{l+(N_\xi-2)} + [\mathcal{G}_l][\Delta\mathbf{U}^n]_{l+(N_\xi-2)(N_\eta-2)} = [\mathbf{RHS}]_l\end{aligned} \tag{4.46}$$

where, $l = i + (N_\xi - 2)(j-1) + (N_\xi - 2)(N_\eta - 2)(k-1)$ and $(i,j,k) \in D_h$. The (5×5) matrices are given by:

$$\begin{aligned}[\mathcal{A}_l] &= -\Delta\tau(-1/2h_\zeta)[\mathbf{C}(\mathbf{U}^n_{i,j,k-1})] - \Delta\tau(1/h_\zeta^2)[\mathbf{S}(\mathbf{U}^n_{i,j,k-1})], \\ [\mathcal{B}_l] &= -\Delta\tau(-1/2h_\eta)[\mathbf{B}(\mathbf{U}^n_{i,j-1,k})] - \Delta\tau(1/h_\eta^2)[\mathbf{Q}(\mathbf{U}^n_{i,j-1,k})], \\ [\mathcal{C}_l] &= -\Delta\tau(-1/2h_\xi)[\mathbf{A}(\mathbf{U}^n_{i-1,j,k})] - \Delta\tau(1/h_\xi^2)[\mathbf{N}(\mathbf{U}^n_{i-1,j,k})], \\ [\mathcal{D}_l] &= \mathbf{I} + \Delta\tau(2/h_\xi^2)[\mathbf{N}(\mathbf{U}^n_{i,j,k})] + \Delta\tau(2/h_\eta^2)[\mathbf{Q}(\mathbf{U}^n_{i,j,k})] + \Delta\tau(2/h_\zeta^2)[\mathbf{S}(\mathbf{U}^n_{i,j,k})], \\ [\mathcal{E}_l] &= -\Delta\tau(1/2h_\xi)[\mathbf{A}(\mathbf{U}^n_{i+1,j,k})] - \Delta\tau(1/h_\xi^2)[\mathbf{N}(\mathbf{U}^n_{i+1,j,k})], \\ [\mathcal{F}_l] &= -\Delta\tau(1/2h_\eta)[\mathbf{B}(\mathbf{U}^n_{i,j+1,k})] - \Delta\tau(1/h_\eta^2)[\mathbf{Q}(\mathbf{U}^n_{i,j+1,k})], \\ [\mathcal{G}_l] &= -\Delta\tau(1/2h_\zeta)[\mathbf{C}(\mathbf{U}^n_{i,j,k+1})] - \Delta\tau(1/h_\zeta^2)[\mathbf{S}(\mathbf{U}^n_{i,j,k+1})].\end{aligned} \tag{4.47}$$

Also, $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ are the elements in the l-th block row of the matrix $\mathbf{Y}$, whereas $\mathcal{E}$,$\mathcal{F}$ and $\mathcal{G}$ are the elements in the l-the block row of the matrix $\mathbf{Z}$.

5. BENCHMARKS

There are three benchmarks associated with the three numerical schemes described in Sec.(4), for the solution of system of linear equations given by Eq.(4.25). Each benchmark consists of running one of the three numerical schemes for N_s time steps with given values for N_ξ, N_η, N_ζ and $\Delta\tau$.

5.1. VERIFICATION TEST

For each of the benchmarks, at the completion of N_s time steps, compute the following:

1) Root Mean Square norms $RMSR(m)$, of the residual vectors $[\mathbf{RHS}^{(m)}]^n_{i,j,k}$, for $m = 1, 2, \ldots, 5$ where $n = N_s$, and $(i,j,k) \in D_h$.

$$RMSR(m) = \sqrt{\frac{\sum_{k=2}^{(N_\zeta-1)}\sum_{j=2}^{(N_\eta-1)}\sum_{i=2}^{(N_\xi-1)}\{[\mathbf{RHS}^{(m)}]^{N_s}_{(i,j,k)}\}^2}{(N_\xi-2)(N_\eta-2)(N_\zeta-2)}}. \tag{5.1}$$

2) Root Mean Square norms $RMSE(m)$ of the error vectors $\{[\mathrm{U}^*]^{(m)}_{i,j,k} - [\mathrm{U}^n]^{(m)}_{i,j,k}\}$, for $m = 1, 2, \ldots, 5$ where $n = N_s$, and $(i,j,k) \in D_h$.

$$RMSE(m) = \sqrt{\frac{\sum_{k=2}^{(N_\zeta-1)} \sum_{j=2}^{(N_\eta-1)} \sum_{i=2}^{(N_\xi-1)} \{[\mathrm{U}^*]^{(m)}_{(i,j,k)} - [\mathrm{U}^{N_s}]^{(m)}_{(i,j,k)}\}^2}{(N_\xi - 2)(N_\eta - 2)(N_\zeta - 2)}}. \tag{5.2}$$

3) The numerically evaluated surface integral given by:

$$\begin{aligned} I = 0.25\{ & \sum_{j=j_1}^{j_2-1} \sum_{i=i_1}^{i_2-1} h_\xi h_\eta [\varphi_{i,j,k_1} + \varphi_{i+1,j,k_1} + \varphi_{i,j+1,k_1} + \varphi_{i+1,j+1,k_1} \\ & + \varphi_{i,j,k_2} + \varphi_{i+1,j,k_2} + \varphi_{i,j+1,k_2} + \varphi_{i+1,j+1,k_2}] \\ & + \sum_{k=k_1}^{k_2-1} \sum_{i=i_1}^{i_2-1} h_\xi h_\zeta [\varphi_{i,j_1,k} + \varphi_{i+1,j_1,k} + \varphi_{i,j_1,k+1} + \varphi_{i+1,j_1,k+1} \\ & + \varphi_{i,j_2,k} + \varphi_{i+1,j_2,k} + \varphi_{i,j_2,k+1} + \varphi_{i+1,j_2,k+1}] \\ & + \sum_{k=k_1}^{k_2-1} \sum_{j=j_1}^{j_2-1} h_\eta h_\zeta [\varphi_{i_1,j,k} + \varphi_{i_1,j+1,k} + \varphi_{i_1,j,k+1} + \varphi_{i_1,j+1,k+1} \\ & + \varphi_{i_2,j,k} + \varphi_{i_2,j+1,k} + \varphi_{i_2,j,k+1} + \varphi_{i_2,j+1,k+1}]\}, \end{aligned} \tag{5.3}$$

where, i_1, i_2, j_1, j_2, k_1 and k_2 are specified constants, such that $1 < i_1 < i_2 < N_\xi$, $1 < j_1 < j_2 < N_\eta$ and $1 < k_1 < k_2 < N_\zeta$, and:

$$\varphi = c_2\{u^{(5)} - 0.5(\frac{[u^{(1)}]^2 + [u^{(2)}]^2 + [u^{(3)}]^2)}{u^{(1)}})\}. \tag{5.4}$$

The validity of each these quantities is measured according to:

$$\frac{|X_c - X_r|}{|X_r|} \leq \epsilon,$$

where X_c and X_r are the computed and reference values, respectively, and ϵ is the maximum allowable relative error. The value of ϵ is specified to be 10^{-8}. The values of X_r are dependent on the numerical scheme under consideration and the values specified for $N_\xi, N_\eta, N_\zeta, \Delta\tau$ and N_s.

5.2. BENCHMARK 1

Perform $N_s = 200$ iterations of the Approximate Factorization Algorithm, with the following parameter values:

$$N_\xi = 64; \qquad N_\eta = 64; \qquad N_\zeta = 64,$$

and

$$\Delta\tau = 0.0008.$$

Timing for this benchmark should begin just before the STEP 1 of the first iteration is started and end just after the STEP 5 of the N_s-th iteration is complete.

5.3. BENCHMARK 2

Perform $N_s = 400$ iterations of the Diagonal Form of the Approximate Factorization Algorithm, with the following parameter values:

$$N_\xi = 64; \qquad N_\eta = 64; \qquad N_\zeta = 64,$$

and

$$\Delta\tau = 0.0015.$$

Timing for this benchmark should begin just before the STEP 1 of the first iteration is started and end just after the STEP 9 of the N_s-th iteration is complete.

5.4. BENCHMARK 3

Perform $N_s = 250$ iterations of the Symmetric Successive Over-Relaxation Algorithm with the following parameter values:

$$N_\xi = 64; \qquad N_\eta = 64; \qquad N_\zeta = 64,$$

and

$$\Delta\tau = 2.0 \qquad \omega = 1.2.$$

Timing for this benchmark should begin just before the STEP 1 of the first iteration is started and end just after the STEP 4 of the N_s-th iteration is complete.

For all benchmarks, values of the remaining constants are specified as:

$$k_1 = 1.40; \quad k_2 = 0.40; \quad k_3 = 0.10; \quad k_4 = 1.00; \quad k_5 = 1.40$$

$$C_{1,1} = 2.0; \quad C_{2,1} = 1.0; \quad C_{3,1} = 2.0; \quad C_{4,1} = 2.0; \quad C_{5,1} = 5.0$$

$$C_{1,2} = 0.0; \quad C_{2,2} = 0.0; \quad C_{3,2} = 2.0; \quad C_{4,2} = 2.0; \quad C_{5,2} = 4.0$$

$$C_{1,3} = 0.0; \quad C_{2,3} = 0.0; \quad C_{3,3} = 0.0; \quad C_{4,3} = 0.0; \quad C_{5,3} = 3.0$$

$$C_{1,4} = 4.0; \quad C_{2,4} = 0.0; \quad C_{3,4} = 0.0; \quad C_{4,4} = 0.0; \quad C_{5,4} = 2.0$$

$$C_{1,5} = 5.0; \quad C_{2,5} = 1.0; \quad C_{3,5} = 0.0; \quad C_{4,5} = 0.0; \quad C_{5,5} = 0.1$$

$$C_{1,6} = 3.0; \quad C_{2,6} = 2.0; \quad C_{3,6} = 2.0; \quad C_{4,6} = 2.0; \quad C_{5,6} = 0.4$$

$$C_{1,7} = 0.5; \quad C_{2,7} = 3.0; \quad C_{3,7} = 3.0; \quad C_{4,7} = 3.0; \quad C_{5,7} = 0.3$$

$$C_{1,8} = 0.02; \quad C_{2,8} = 0.01; \quad C_{3,8} = 0.04; \quad C_{4,8} = 0.03; \quad C_{5,8} = 0.05$$

$$C_{1,9} = 0.01; \quad C_{2,9} = 0.03; \quad C_{3,9} = 0.03; \quad C_{4,9} = 0.05; \quad C_{5,9} = 0.04$$

$$C_{1,10} = 0.03; \quad C_{2,10} = 0.02; \quad C_{3,10} = 0.05; \quad C_{4,10} = 0.04; \quad C_{5,10} = 0.03$$

$$C_{1,11} = 0.5; \quad C_{2,11} = 0.4; \quad C_{3,11} = 0.3; \quad C_{4,11} = 0.2; \quad C_{5,11} = 0.1$$

$$C_{1,12} = 0.4; \quad C_{2,12} = 0.3; \quad C_{3,12} = 0.5; \quad C_{4,12} = 0.1; \quad C_{5,12} = 0.3$$

$$C_{1,13} = 0.3; \quad C_{2,13} = 0.5; \quad C_{3,13} = 0.4; \quad C_{4,13} = 0.3; \quad C_{5,13} = 0.2$$

$$d_\xi^{(1)} = d_\xi^{(2)} = d_\xi^{(3)} = d_\xi^{(4)} = d_\xi^{(5)} = 0.75$$

$$d_\eta^{(1)} = d_\eta^{(2)} = d_\eta^{(3)} = d_\eta^{(4)} = d_\eta^{(5)} = 0.75$$

$$d_\zeta^{(1)} = d_\zeta^{(2)} = d_\zeta^{(3)} = d_\zeta^{(4)} = d_\zeta^{(5)} = 1.00$$

$$\varepsilon = [max(d_\xi^{(1)}, d_\eta^{(1)}, d_\zeta^{(1)})]/4.0$$

REFERENCES

1. Bailey, D. H., and Barton, J.T.,*The NAS Kernel Benchmark Program, NASA Technical Memorandum 86711, NASA/Ames, Aug. 1985.*

2. Pulliam, T.H.,*Efficient Solution Methods for the Navier-Stokes Equations*, Lecture Notes for the Von Karman Institute for Fluid Dynamics Lecture Series: Numerical Techniques for Viscous Flow Computation in Turbomachinery Bladings, Jan. 20-24, 1986, Brussels, Belgium.

3. Yoon. S., Kwak, D. and Chang, L.,*LU-SGS implicit algorithm for three-dimensional incompressible Navier-Stokes equations with source term*, AIAA Paper 89-1964-CP, 1989.

4. Jameson, A., Schmidt, W. and Turkel, E.,*Numerical solution of the Euler Equations by Finite Volume Methods using Runge-Kutta Time-stepping Schemes*, AIAA Paper, 81-1259, June 1981.

5. Steger, J.L. and Warming, R.F.,*Flux Vector Splitting of the Inviscid Gas Dynamics Equations with Applications to Finite Difference Methods*, Journal of Computational Physics, Vol. 40, (1981), p. 263.

6. van Leer, B.,*Flux Vector Splitting for the Euler Equations*, Eigth International Conference on Numerical Methods in Fluid Dynamics, Lecture Notes in Physics, Vol. 170, Ed. E. Krause, 1982, pp. 507-512.

7. Roe, P.L.,*Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes*, Journal of Computational Physics, Vol. 43, 1981, pp. 357-372.

8. Harten, A. *High Resolution Schemes for Hyperbolic Conservation Laws*, Journal of Computational Physiscs, Vol. 49, 1983, pp. 357-393.

9. Gordon, W.J.,*Blending function methods for bivariate and multivariate interpolation*, SIAM J. Num. Analysis, 8(1971), pp. 158.

10. Warming, R.F. and Beam, R.M.,*On the construction and application of implicit factored schemes for conservation Laws*, Symposium on Computational Fluid Dynamics, SIAM-AMS Proceedings, Vol. 11, 1978.

11. Lapidus, L. and Seinfeld, J.H.,*Numerical Solution of Ordinary Differential Equations*, Academic Press, New York, 1971.

12. Lomax, H.,*Stable implicit and explicit numerical methods for integrating quasi-linear differential equations with parasitic-stiff and parasitic-saddle eigenvalues*, NASA TN D-4703, 1968.

13. Pulliam, T.H. and Chaussee, D.S.,*A Diagonal Form of an Implicit Approximate Factorization Algorithm*, Journal of Computational Physics, Vol. 39, (1981), p. 347.

14. Chan, T.F. and Jackson, K.R.,*Nonlinearly Preconditioned Krylov Subspace Methods for Discrete Newton Algorithms*, SIAM J. Sci. Stat. Compt. Vol. 5, (1984), pp. 533-542.

15. Axelsson, O.,*A generalized SSOR method*, BIT, Vol.13, (1972), pp. 443-467.

16. Olsson, P. and Johnson, S.L.,*Boundary Modifications of the Dissipation Operators for the Three-Dimensional Euler Equations*, Journal of Scientific Computing, Vol. 4, (1989), pp. 159-195.

17. Coakley, T.J.,*Numerical Methods for Gas Dynamics Combining Characteristics and Conservation Concepts*, AIAA Paper 81-1257 (1981).